A Quick Survey on Deep Learning Engines

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

Outline

Introduction

Comparison

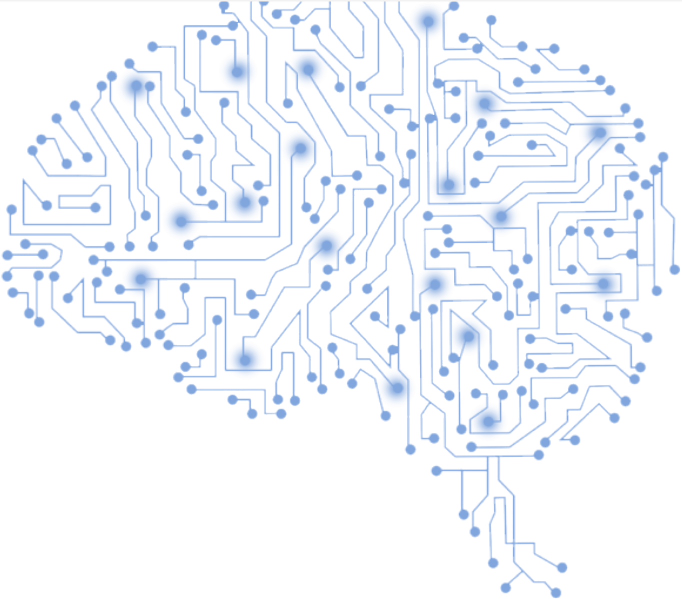
Python

TensorFlow

MXNet

Torch

Caffe



§1. Introduction

1 Machine Learning

2 ML Software Packages

3 Deep Learning

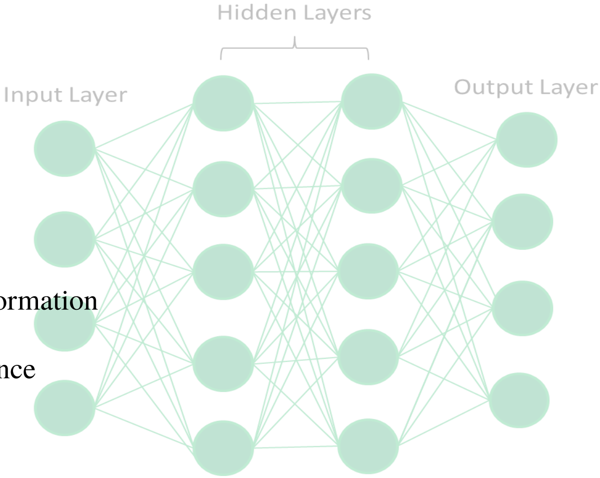
4 Goal of This Study

5 DL Engines: Basic Information

6 DL Engines: Performance

7 Hardware Platforms

8 Neural Networks and Test Data Sets



Machine Learning

What is ML?

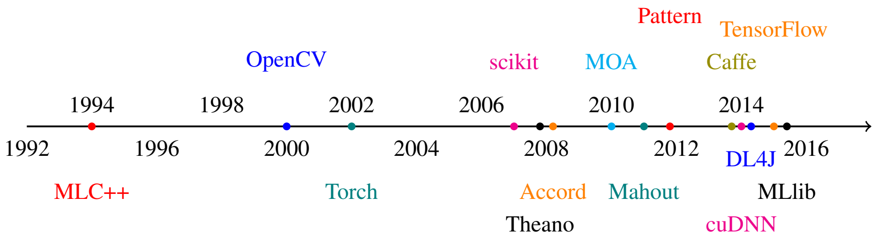
- Unlike traditional numerical simulation, "ML gives computers the ability to learn without being explicitly programmed" [Samuel 1959]
- As a research field, ML explores the study and construction of algorithms that can learn from and make predictions on data
- Fourth paradigm, big data, Internet of 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
- Clustering: Inputs are divided into several groups. Unlike in classification, the groups are not known beforehand, making this an unsupervised task
- Regression: Similar to classification, but the outputs are continuous
- Density estimation, dimensionality reduction, ...

I. Goodfellow, Y. Bengio, and A. Courville, https://github.com/exacity/deeplearningbook-chinese

ML Software Packages



☞ Other worth-noting packages:

- CNTK/DMTK (Microsoft):
  - Support Windows/Linux, no official OS X support
  - Provide C++/Python frontend
- Neon (Nervana & Intel)
- Caffe2 (Facebook)
- PyTorch (beta)

Tour of TensorFlow, by Peter Goldsborough, arXiv, 2016

Deep Learning

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

Goal of This Study

What is our purpose?

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

How to use a ML package?

- Train models on large data set on high-performance computing platforms
- Deploy applications on various computing platforms

What we want for a ML package?

- Easy to build new tasks and new network structures (less steep learning curve)
- Easy for debugging (with good documentation, support, and large community)
- Good performance and scalability
  - Multicore CPU
  - Single or multiple GPU(s)
  - Cluster

DL Engines: Basic Information

Viewpoint	Torch	Caffe	TensorFlow	MXNet
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	C++/Python/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

- 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 980	2048	4 GB	Ubuntu 14.04	8.0
GTX 1080	2560	8 GB	Ubuntu 14.04	8.0
Telsa K80 GK210	2496	12 GB	CentOS 7.2	8.0

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

9 CPU Scalability: FCN Synthetic

10 CPU Scalability: FCN Real

11 CPU Scalability: CNN Synthetic

12 CPU Scalability: CNN Real

13 CPU Scalability: RNN LSTM

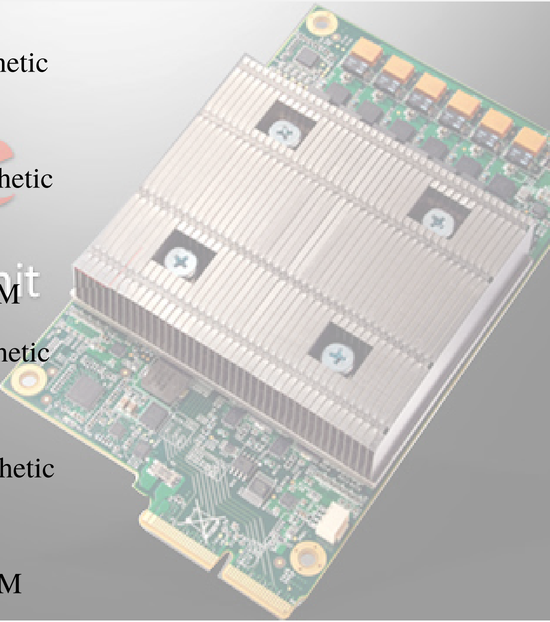
14 GPU Scalability: FCN Synthetic

15 GPU Scalability: FCN Real

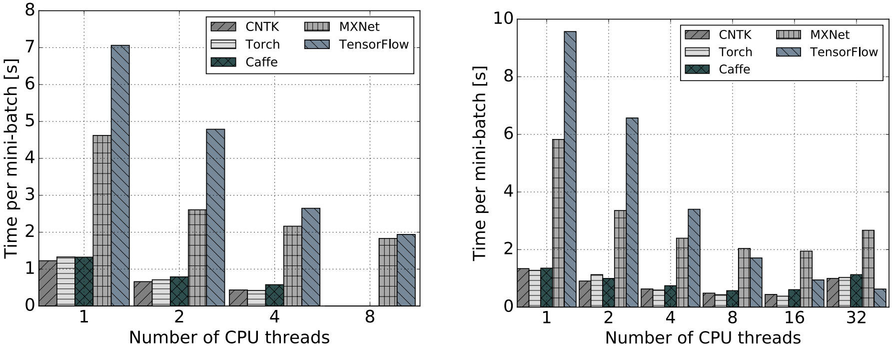
16 GPU Scalability: CNN Synthetic

17 GPU Scalability: CNN Real

18 GPU Scalability: RNN LSTM



CPU Scalability: FCN Synthetic



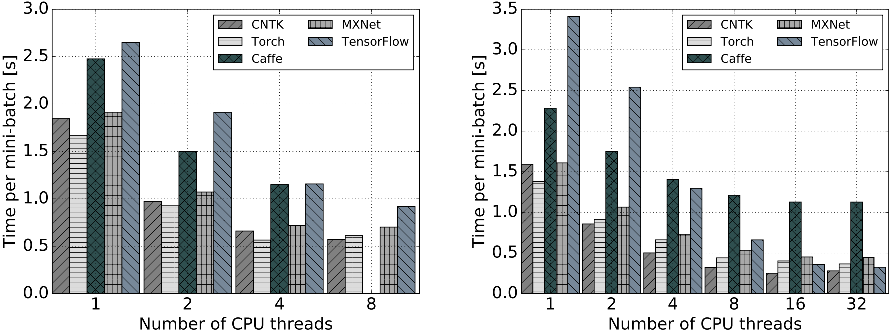
(a) Results on i7-3820.

(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

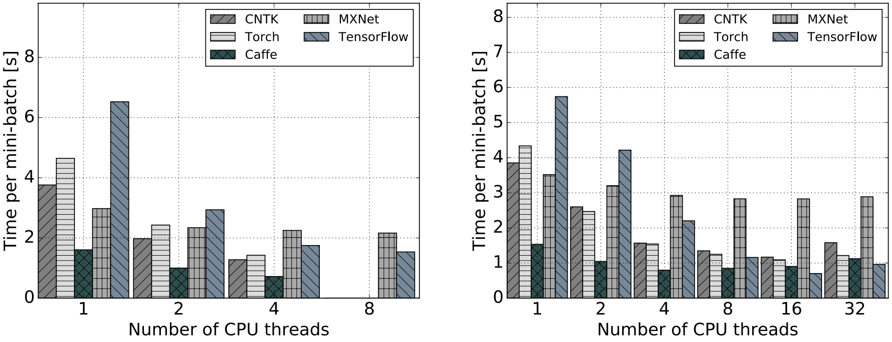


(a) Results on i7-3820. (b) Results on E5-2630.

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

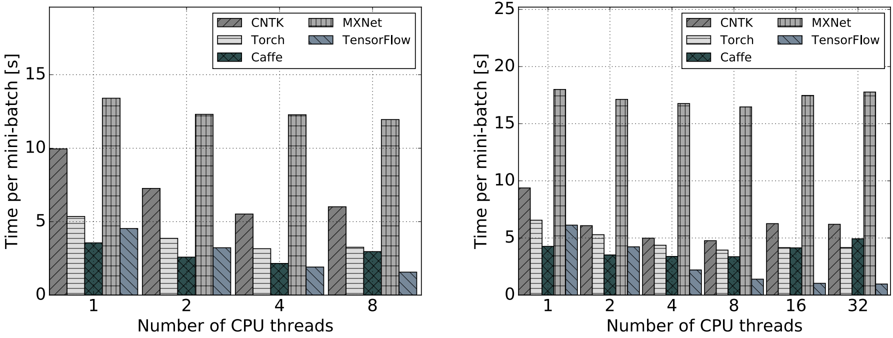


(a) Results on i7-3820. (b) Results on E5-2630.

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

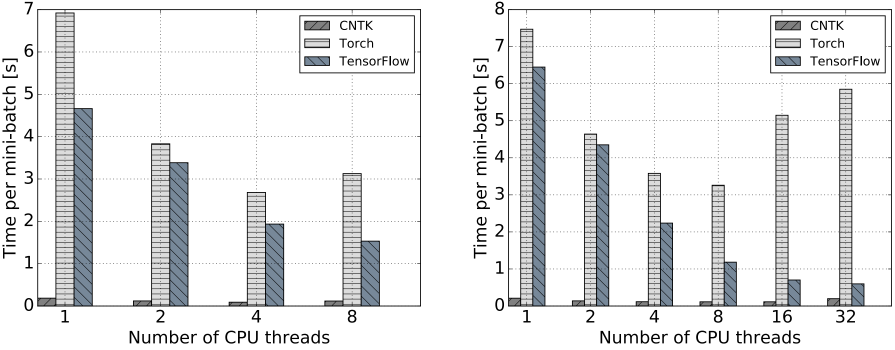


(a) Results on i7-3820. (b) Results on E5-2630.

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

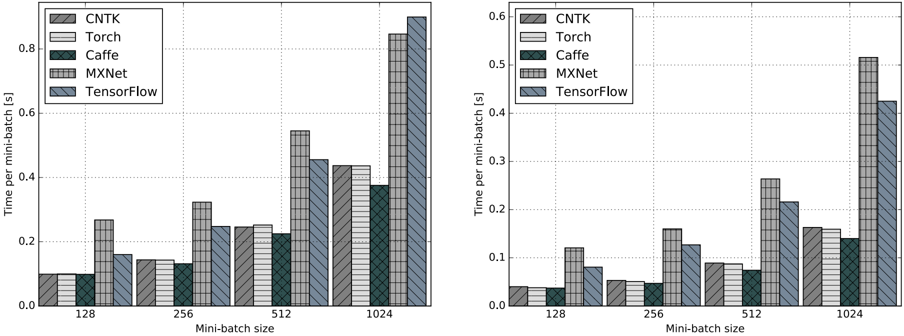


(a) Results on i7-3820. (b) Results on E5-2630.

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 Scalability: FCN Synthetic

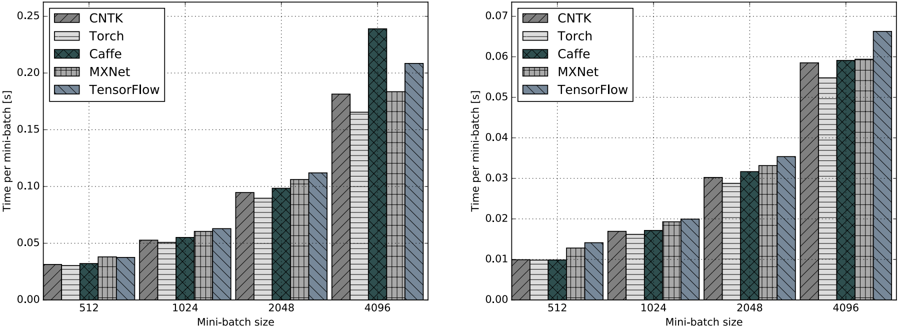


(a) Results on Tesla K80. (b) Results on GTX1080.

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

- CNTK/Torch/Caffe out-perform the others

GPU Scalability: FCN Real

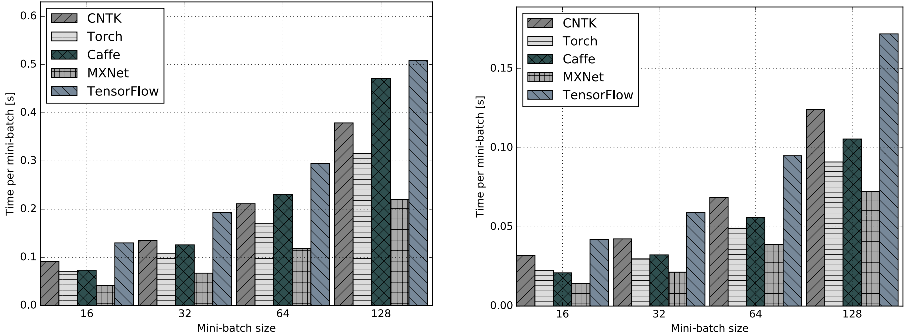


(a) Results on Tesla K80. (b) Results on GTX1080.

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

- All packages have similar performance

GPU Scalability: CNN Synthetic

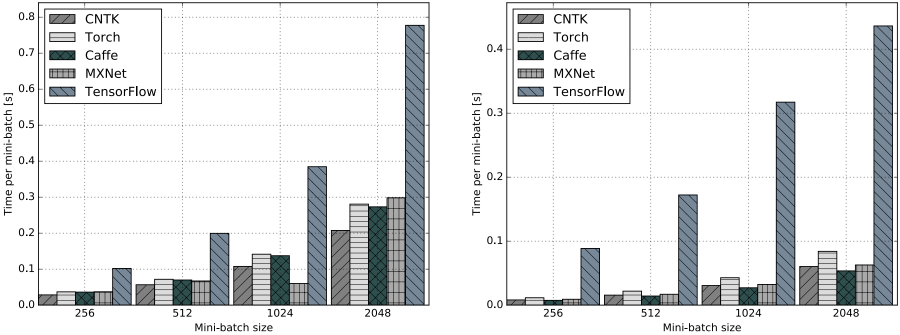


(a) Results on Tesla K80. (b) Results on GTX1080.

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

- MXNet out-perform the others for CNN on GPUs

GPU Scalability: CNN Real



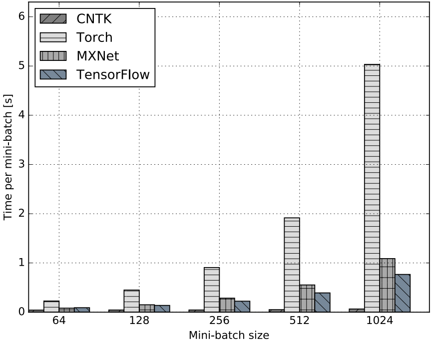
(a) Results on Tesla K80.

(b) Results on GTX1080.

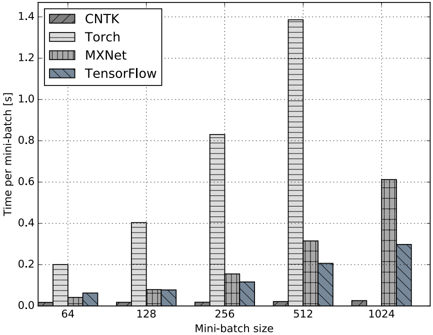
Figure: The performance comparison of AlexNet-R on GPU platforms

- TensorFlow does not have good GPU performance in general

GPU Scalability: RNN LSTM



(a) Results on Tesla K80.



(b) Results on GTX1080.

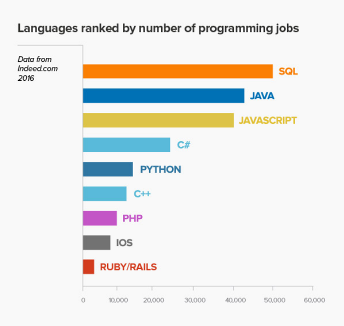
Figure: The performance comparison of LSTM on GPU platforms

- CNTK has excellent RNN performance both on CPU and GPU

§3. Python

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

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().

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], (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

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
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 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.

```
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/Pylab: for visualising data
   - SciPy: providing lots of numerical algorithms
   - SymPy: for symbolic mathematics

http://www.pythontutor.com

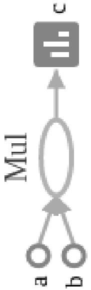
§4. TensorFlow

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.

Computational Graph

```python
import tensorflow as tf

graph = tf.Graph()

with graph.as_default():
    # Define two constants
    a = tf.constant(1.0)
    b = tf.constant(2.0)

    # Define an operation node: c = a * b
    c = tf.multiply(a, b)
    # Add scalar summary to operation node
    tf.summary.scalar('c', c)

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

with tf.Session(graph=graph):
    # Write data to ./board
    writer.add_summary(merged.eval())
```



Example: SoftMax in TF

```
import tensorflow as tf

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

# Create a new TensorFlow graph
graph = tf.Graph()
with graph.as_default():
    # Nodes or entire subgraphs can be grouped into one visual block for tensorboard
    with tf.name_scope('input_features'):
        # Placeholder for input variables (None = variable dimension)
        x = tf.placeholder(tf.float32, shape=[None, 784], name='input_x')

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

    with tf.name_scope('variables'):
        W = tf.Variable(tf.zeros([784, 10]), name='weights')
        # Track tensor distributions over time for tensorboard
        tf.summary.histogram('WEIGHTS', W)
        b = tf.Variable(tf.zeros([10]), name='biases')
        tf.summary.histogram('BIASES', b)

    with tf.name_scope('use_softmax'):
        # Apply softmax regression model to the input data and get prediction y
        y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Example: SoftMax in TF (Cont)

```
    with tf.name_scope('train'):
        # Compute the cross entropy of real label y_ and prediction labe 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)

    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 tensor values over time for tensorboard
        tf.summary.scalar('Accuracy', accuracy)

    merged = tf.summary.merge_all()   # Merge all the summaries
    writer = tf.summary.FileWriter('./board', graph)  # Write summary file to ./board

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

    for step in range(1000):
        if (step%10) == 0:
            # Feed test data to compute accuarcy
            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:
            # Feed training data to train the model
            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}),
                               global_step=step)
```

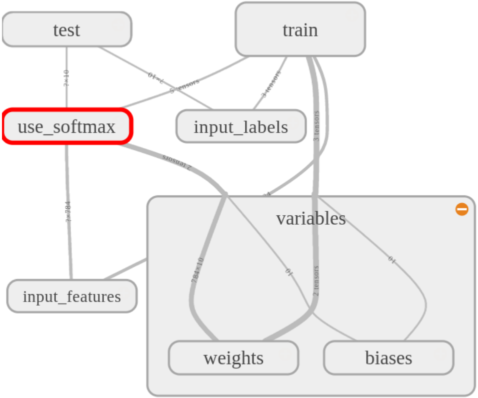
Visualization: TensorBoard

Declarative style

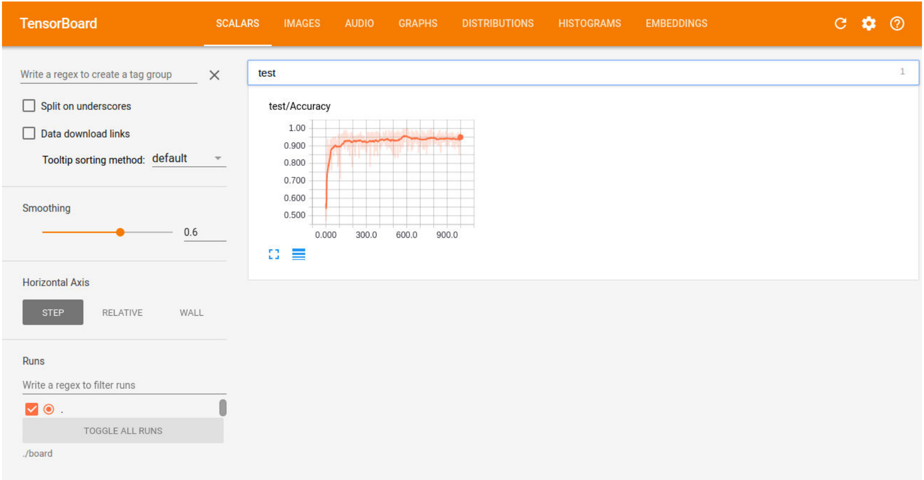
- Same style as Teano
- Easy to understand
- Possible for graph optimization
- More difficult for debugging

Computation graphs are powerful but complicated

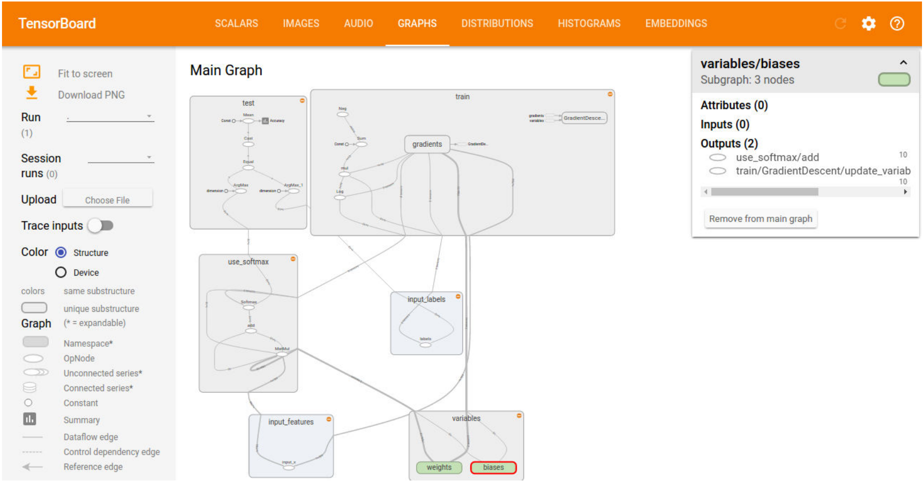
- Thousands of nodes or more
- Network is deep
- Visualization is helpful for debugging



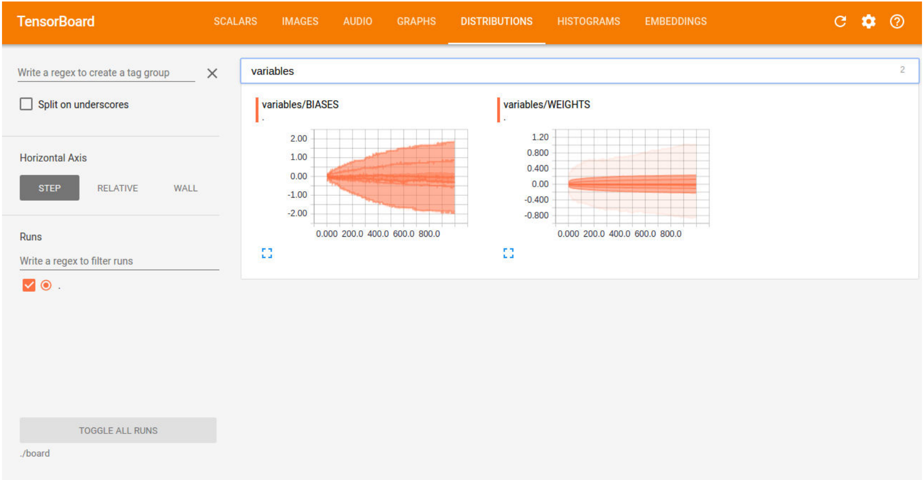
TensorBoard: Scalars



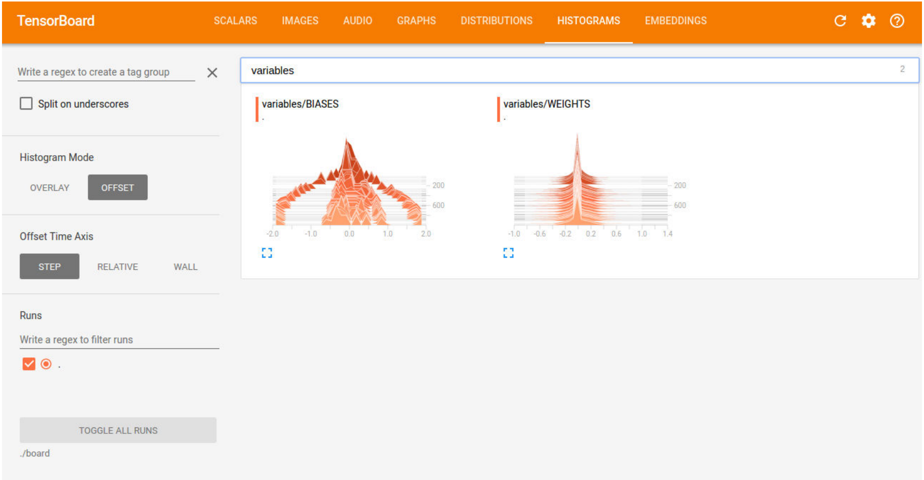
TensorBoard: Graphs



TensorBoard: Distributions



TensorBoard: Histograms



§5. MXNet

Programming Interface

1. Support many scope applications (e.g. computer vision, natural language processing, speech recognition, unsupervised machine learning, support embedded APIs, visualization)
2. Mixed programming style: imperative and declarative
   - Data parallelism with multi-devices: Better scalability than TensorFlow, reportedly
   - Support many different front-end, including JavaScript (so it be run on web browsers as well)
   - Provide intermediate-level and high-level interface modules
   - Provide abundant IO functions
3. Fully compatible with Torch: modules and operators
4. Support building neural network graphs
   - Call mx.viz.plot_network( )
5. Not well documented

Example: SVM in MXNet

```
from __future__ import print_function
import mxnet as mx
import numpy as np
from sklearn.datasets import fetch_mldata
from sklearn.decomposition import PCA
# import matplotlib.pyplot as plt
import logging

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

# Network declaration as symbols. The following pattern was based
# on the article, but feel free to play with the number of nodes
# and with the activation function
data = mx.symbol.Variable('data')
fc1  = mx.symbol.FullyConnected(data = data, name='fc1', num_hidden=512)
act1 = mx.symbol.Activation(data = fc1, name='relu1', act_type="relu")
fc2  = mx.symbol.FullyConnected(data = act1, name = 'fc2', num_hidden = 512)
act2 = mx.symbol.Activation(data = fc2, name='relu2', act_type="relu")
fc3  = mx.symbol.FullyConnected(data = act2, name='fc3', num_hidden=10)

# Here we add the ultimate layer based on L2-SVM objective
mlp = mx.symbol.SVMOutput(data=fc3, name='svm')

# To use L1-SVM objective, comment the line above and uncomment the line below
# mlp = mx.symbol.SVMOutput(data=fc3, name='svm', use_linear=True)

# Now we fetch MNIST dataset, add some noise, as the article suggests,
# permutate and assign the examples to be used on our network
mnist = fetch_mldata('MNIST original')
```

Example: SVM in MXNet (Cont)

```
mnist_pca = PCA(n_components=70).fit_transform(mnist.data)
noise = np.random.normal(size=mnist_pca.shape)
mnist_pca += noise

# set seed for deterministic ordering
np.random.seed(1234)
p = np.random.permutation(mnist_pca.shape[0])
X = mnist_pca[p]
Y = mnist.target[p]
X_show = mnist.data[p]

# This is just to normalize the input to a value inside [0,1],
# and separate train set and test set
X = X.astype(np.float32)/255
X_train = X[:60000]
X_test = X[60000:]
X_show = X_show[60000:]
Y_train = Y[:60000]
Y_test = Y[60000:]

# Article's suggestion on batch size
batch_size = 200
train_iter = mx.io.NDArrayIter(X_train, Y_train, batch_size=batch_size)
test_iter = mx.io.NDArrayIter(X_test, Y_test, batch_size=batch_size)

# A quick work around to prevent mxnet complaining the lack of a softmax_label
train_iter.label= mx.io._init_data(Y_train,allow_empty=True,default_name='svm_label')
test_iter.label= mx.io._init_data(Y_test,allow_empty=True,default_name='svm_label')
```

Example: SVM in MXNet (Cont)

```
# Here we instatiate and fit the model for our data
# The article actually suggests using 400 epochs,
# But I reduced to 10, for convinience
model = mx.model.FeedForward(
    ctx = mx.cpu(0),          # Run on CPU 0
    symbol = mlp,             # Use the network we just defined
    num_epoch = 10,           # Train for 10 epochs
    learning_rate = 0.1,      # Learning rate
    momentum = 0.9,           # Momentum for SGD with momentum
    wd = 0.00001,             # Weight decay for regularization
    )

model.fit(
    # Training data set
    X=train_iter,
    # Testing data set. MXNet computes scores on test set every epoch
    eval_data=test_iter,
    # Logging module to print out progress
    batch_end_callback = mx.callback.Speedometer(batch_size, 200)
    )

# Uncomment to view an example
# plt.imshow((X_show[0].reshape((28,28))*255).astype(np.uint8), cmap='Greys_r')
# plt.show()
# print 'Result:', model.predict(X_test[0:1])[0].argmax()

# Now it prints how good did the network did for this configuration
print('Accuracy:', model.score(test_iter)*100, '%')
```

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

§6. Torch

Programming Interface

1. Wide range of applications
   - Speech, image and video applications
   - Large-scale machine-learning applications
2. Fastest scripting language Lua is used
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. Easy extensibility
   - Easy to integrate any library into Torch

Example: Two-Layer 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 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)

```
   #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

Caffe

Deep learning framework by BAIR

General Comments

1. Mainly focus on (and well suited for) CNN and image recognition
2. Expressive architecture
    - Define models and optimization by configuration without hard-coding
    - With protocol tool to define parameters for nets and solvers . . .
3. Not well documented
4. Lots of dependencies; can be tricky to install
5. No automatic differentiation
6. Not so convenient to extend (write layers in C++ or Python, handwritten CUDA code)

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

```python
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.

    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
```

Example: Extend Layers (Cont)

```
            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


Thank You!