

# Software Packages for Deep Learning

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

# Outline

Introduction

Comparison

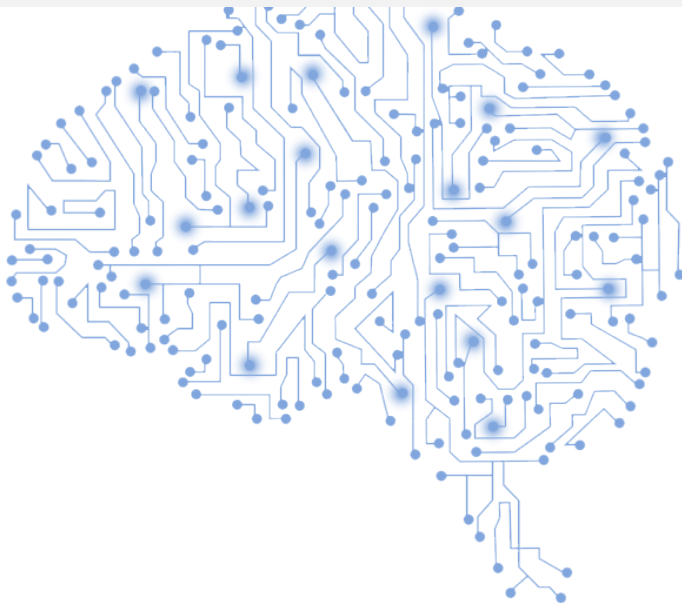
Python

TensorFlow

MXNet

Torch

Caffe



# §1. Introduction

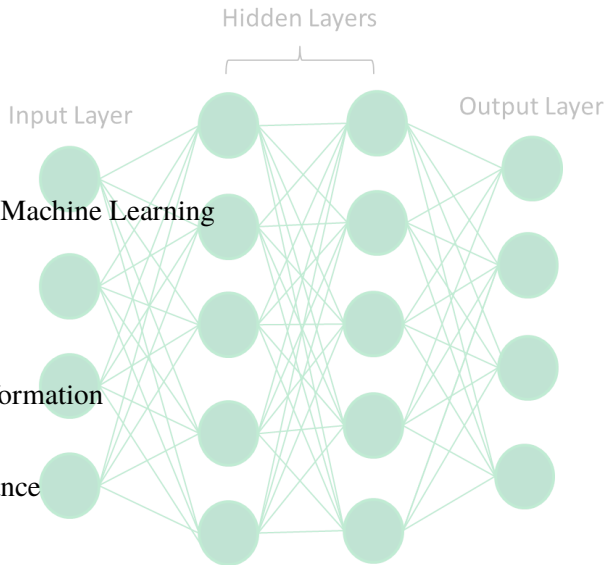
1 Machine Learning

2 Software Packages for Machine Learning

3 Deep Learning

4 DL Engines: Basic Information

5 DL Engines: Performance



# Machine Learning

## Why ML now?

- 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, artificial intelligence, Internet of things, ...

## 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
- Density estimation, dimensionality reduction, ...

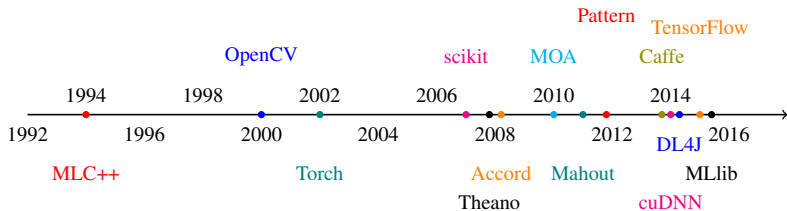
# Software Packages for Machine Learning

## 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

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

# 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++/ <b>Python</b> /R Java/Go	C++/ <b>Python</b> /R Matlab/Julia/...
License	BSD	BSD	Apache	Apache

☞ Other worth-noting's: CNTK/DMTK (Microsoft), Neon (Nervana & Intel), PyTorch (**beta**)

- 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, ...

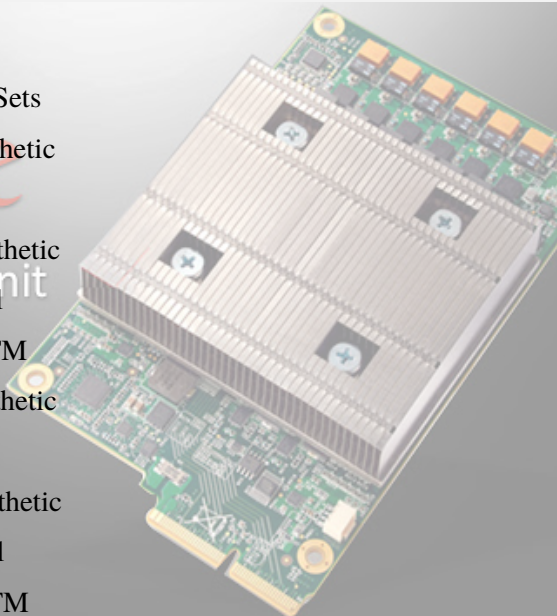
# DL Engines: Performance

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	Increasing fast
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
Coding Style	Imperative	Imperative	Declarative	Declarative Imperative
GitHub Watching	649/268	1856	4939	887



## §2. Comparison

- 6 Hardware Platforms
- 7 Neural Networks and Data Sets
- 8 CPU Scalability: FCN Synthetic
- 9 CPU Scalability: FCN Real
- 10 CPU Scalability: CNN Synthetic
- 11 CPU Scalability: CNN Real
- 12 CPU Scalability: RNN LSTM
- 13 GPU Scalability: FCN Synthetic
- 14 GPU Scalability: FCN Real
- 15 GPU Scalability: CNN Synthetic
- 16 GPU Scalability: CNN Real
- 17 GPU Scalability: RNN LSTM



# Hardware Platforms

- Use one quad-core desktop CPU (i.e., Intel i7-3820 CPU @ 3.60GHz) and two 8-core server-grade CPUs (i.e., Intel Xeon CPU E5-2630 v3 @ 2.40GHz)
- Use two generations of GPU cards, 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

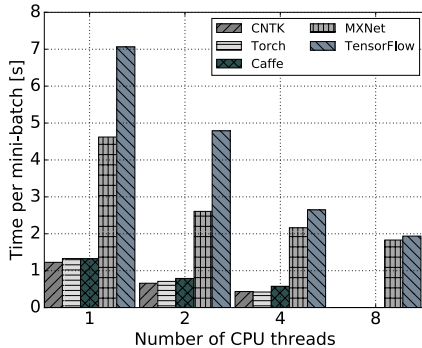
# Neural Networks and Data Sets

- 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 are selected for testing, with input length of 32.

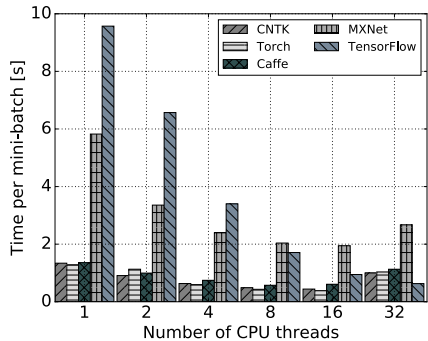
Networks		Input	Output	Layers	Parameters
FCN	FCN-S	26752	26752	5	~55 millions
FCN	FCN-R	784	10	5	~31 millions
CNN	AlexNet-S	150528	1000	4	~61 millions
CNN	AlexNet-R	3072	10	4	~81 thousands
RNN	LSTM	10000	10000	2	~13 millions

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

# CPU Scalability: FCN Synthetic



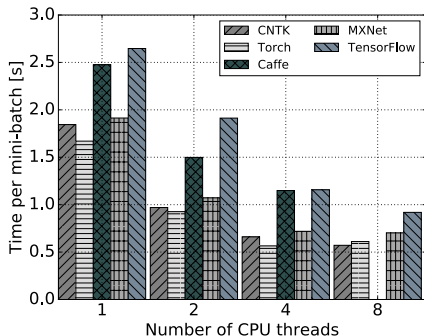
(a) Results on i7-3820.



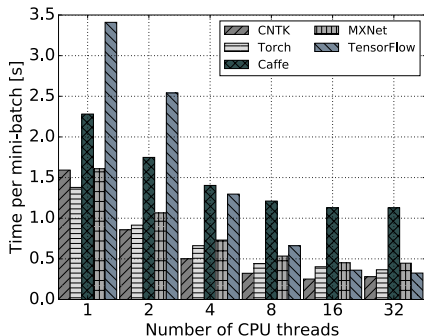
(b) Results on E5-2630.

**Figure:** FCN-S performance comparison on CPU platform with a mini-batch size of 64 (The lower the better)

# CPU Scalability: FCN Real



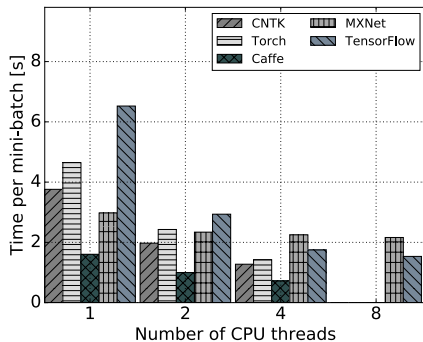
(a) Results on i7-3820.



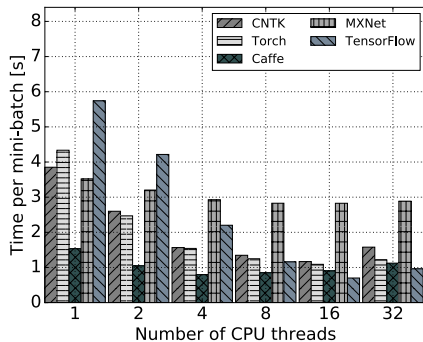
(b) Results on E5-2630.

**Figure:** The FCN-R performance comparison on CPU platform with a mini-batch size of 1024 (The lower the better)

# CPU Scalability: CNN Synthetic



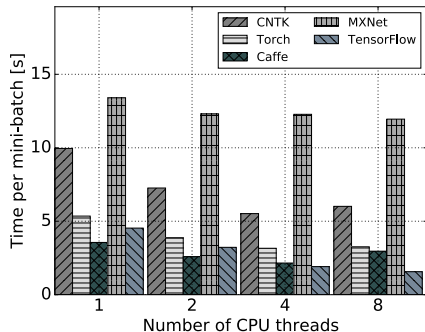
(a) Results on i7-3820.



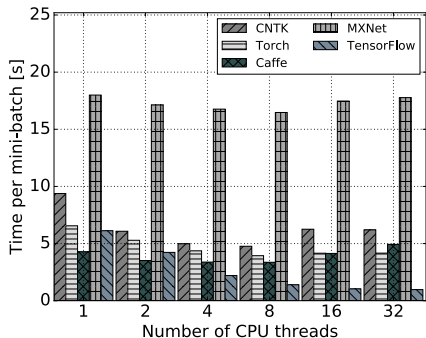
(b) Results on E5-2630.

**Figure:** AlexNet-S performance comparison on CPU platform with a mini-batch size of 16 (The lower the better)

# CPU Scalability: CNN Real



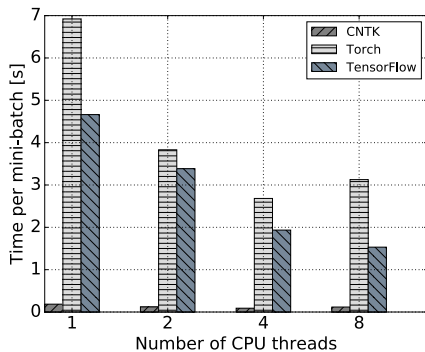
(a) Results on i7-3820.



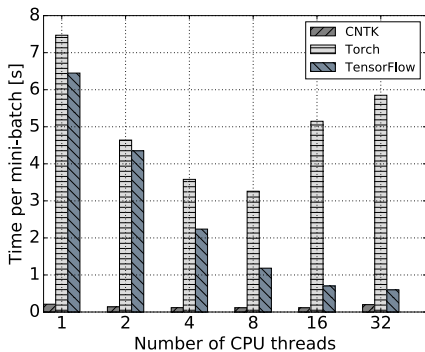
(b) Results on E5-2630.

**Figure:** AlexNet-R performance comparison on CPU platform with a mini-batch size of 1024 (The lower the better)

# CPU Scalability: RNN LSTM



(a) Results on i7-3820.

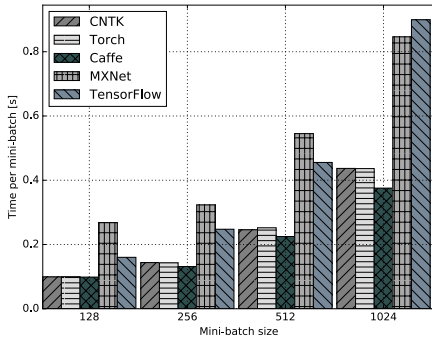


(b) Results on E5-2630.

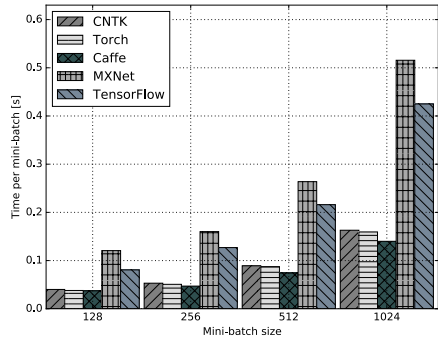
**Figure:** LSTM performance comparison on CPU platform with a mini-batch size of 256 (The lower the better)



# GPU Scalability: FCN Synthetic



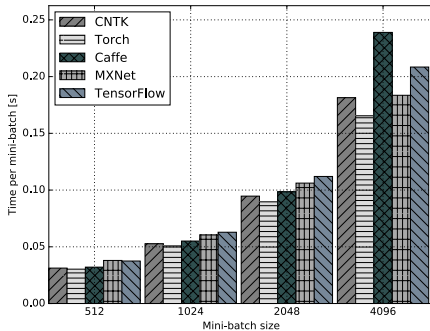
(a) Results on Tesla K80.



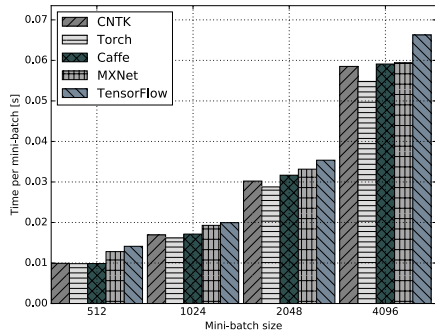
(b) Results on GTX1080.

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

# GPU Scalability: FCN Real



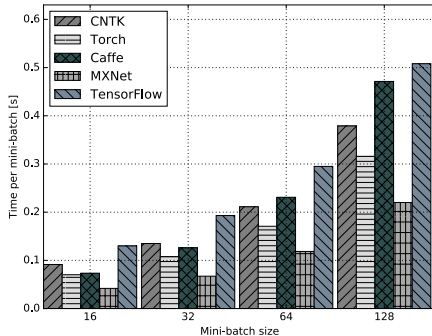
(a) Results on Tesla K80.



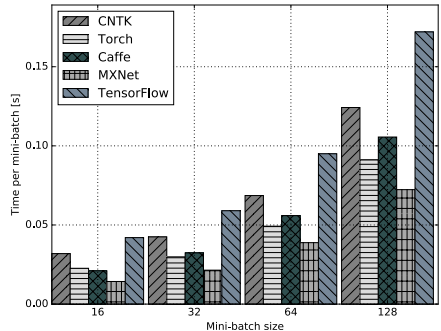
(b) Results on GTX1080.

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

# GPU Scalability: CNN Synthetic



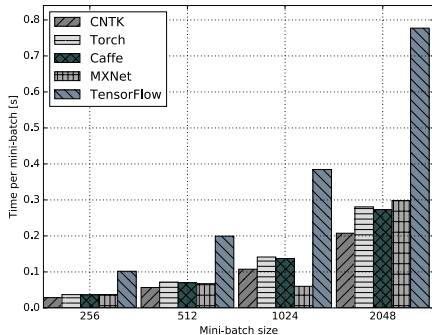
(a) Results on Tesla K80.



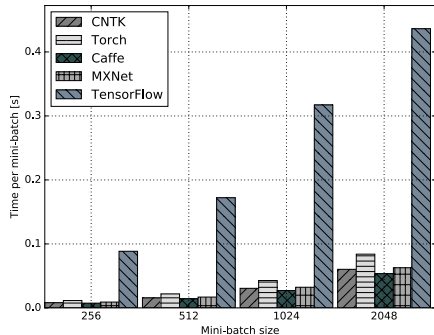
(b) Results on GTX1080.

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

# GPU Scalability: CNN Real



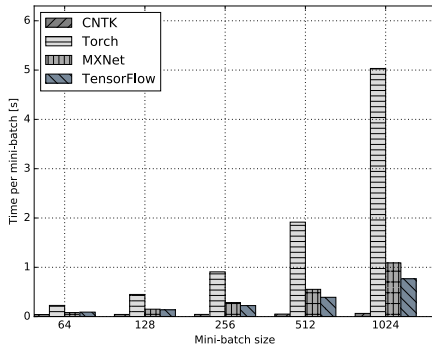
(a) Results on Tesla K80.



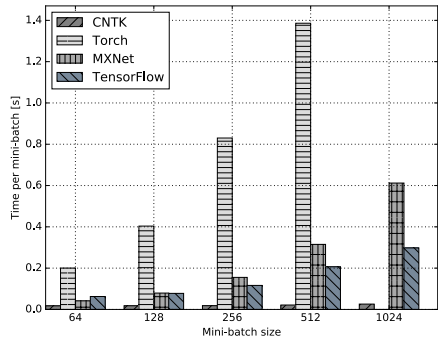
(b) Results on GTX1080.

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

# GPU Scalability: RNN LSTM



(a) Results on Tesla K80.



(b) Results on GTX1080.

**Figure:** The performance comparison of LSTM on GPU platforms

## §3. Python

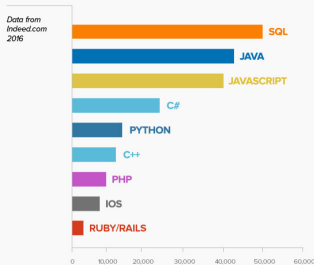
- 18 Python: A general-purpose programming language
- 19 Python for Scientific Computing
- 20 Built-in Data Structures
- 21 Control Flow
- 22 Modules
- 23 Additional Comments



# 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

Languages ranked by number of programming jobs



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<sup>1</sup>
- 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**

---

<sup>1</sup>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

```

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

```

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

```

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

```

## Formatted output

```

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

```

## User defined functions<sup>23</sup>

```

1  def square(x):
2      return x*x

```

<sup>2</sup>Function overhead is high. Not to call a function repeatedly; Using aggregation instead.

<sup>3</sup>Python always passes (the value of) the reference to the object to the function.

# Control Flow

## If-then-else

```
1 a = 1
2 if a > 0:
3     print "a is positive"
4 elif a=0:
5     print "a is zero"
6 else:
7     print "a is negative"
```

## For loop

```
1 # loop from 0 to 9
2 for i in range(10):
3     print i
4
5 # loop over the list named by oldlist
6 newlist = [s.upper() for s in oldlist]
7
8 a = range(5) # create a new list a
9 b = a        # b points to the list a
10 c = [item for item in a] # copy list a to a new list
```

## While loop

```
1 sum = 0; i = 0
2 while i < 10:
3     sum += i
4     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

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

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

- ❷ `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
```

- ❸ Use a profiler to find optimization possibilities

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

- ❹ Some useful and important packages

- NumPy: for scientific computing
- Matplotlib/PyLab: for visualising data
- SciPy: providing lots of numerical algorithms
- SymPy: for symbolic mathematics

## §4. TensorFlow

- 24 Programming Interface
- 25 Computational Graph
- 26 Visualization: TensorBoard
- 27 Example: SoftMax in TF
- 28 Example: SoftMax in TF (Cont)



# Programming Interface

- **Graph:** In TensorFlow, machine learning algorithms are represented as computational graph. A computational or data flow graph is a form of directed graph where vertices or nodes describe operations, while edges represent data flowing between these operations.
- **Operation:** An operation 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.

# Computational Graph

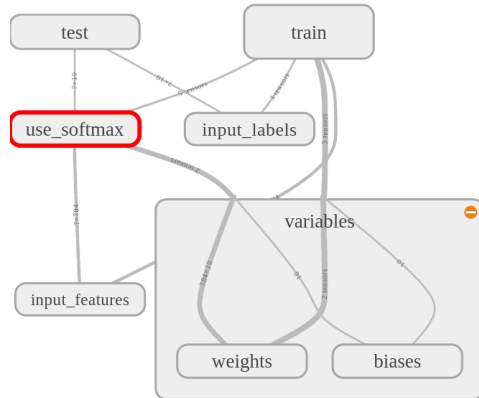
```
1 import tensorflow as tf
2
3 graph = tf.Graph()
4
5 with graph.as_default():
6     # Define two constants
7     a = tf.constant(1.0)
8     b = tf.constant(2.0)
9
10    # Define an operation node: c = a * b
11    c = tf.multiply(a, b)
12    # Add scalar summary to operation node
13    tf.summary.scalar('c', c)
14
15    # Merge all the summaries and write to ./board
16    merged = tf.summary.merge_all()
17    writer = tf.summary.FileWriter('./board', graph)
18
19 with tf.Session(graph=graph):
20     # Write data to ./board
21     writer.add_summary(merged.eval())
```



## Visualization: TensorBoard

## Computation graphs are powerful but complicated

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





# Example: SoftMax in TF

```
1 import tensorflow as tf
2
3 # Import training and test data
4 import tensorflow.examples.tutorials.mnist.input_data as input_data
5 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
6
7 # Create a new TensorFlow graph
8 graph = tf.Graph()
9 with graph.as_default():
10     # Nodes or entire subgraphs can be grouped into one visual block for tensorboard
11     with tf.name_scope('input_features'):
12         # Placeholder for input variables (None = variable dimension)
13         x = tf.placeholder(tf.float32, shape=[None, 784], name='input_x')
14
15     with tf.name_scope('input_labels'):
16         # Placeholder for labels
17         y_ = tf.placeholder(tf.float32, shape=[None, 10], name='labels')
18
19     with tf.name_scope('variables'):
20         W = tf.Variable(tf.zeros([784, 10]), name='weights')
21         # Track tensor distributions over time for tensorboard
22         tf.summary.histogram('WEIGHTS', W)
23         b = tf.Variable(tf.zeros([10]), name='biases')
24         tf.summary.histogram('BIASES', b)
25
26     with tf.name_scope('use_softmax'):
27         # Apply softmax regression model to the input data and get prediction y
28         y = tf.nn.softmax(tf.matmul(x, W) + b)
```

## Example: SoftMax in TF (Cont)

```

29 with tf.name_scope('train'):
30     # Compute the cross entropy of real label y_ and prediction label y
31     cross_entropy = -tf.reduce_sum(y_*tf.log(y))
32     # Create a gradient-descent optimizer with learning rate = 0.01
33     train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
34
35 with tf.name_scope('test'):
36     correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
37     accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
38     # Track tensor values over time for tensorboard
39     tf.summary.scalar('Accuracy', accuracy)
40
41 # Merge all the summaries and write to ./board
42 merged = tf.summary.merge_all()
43 writer = tf.summary.FileWriter('./board', graph)
44
45 with tf.Session(graph=graph) as sess:
46     # Initialize all variables
47     tf.global_variables_initializer().run()
48
49     for step in range(1000):
50         if (step%10) == 0:
51             # Feed test data to compute accuracy
52             feed = {x: mnist.test.images, y_: mnist.test.labels}
53             _, acc = sess.run([merged, accuracy], feed_dict=feed)
54             print('Accuracy at %s step: %s' % (step, acc))
55         else:
56             # Feed training data to train the model
57             batch_x, batch_y = mnist.train.next_batch(100)
58             sess.run(train_step, feed_dict={x: batch_x, y_: batch_y})
59             writer.add_summary(merged.eval(feed_dict={x: batch_x, y_: batch_y}),
60                               global_step=step)

```

## §5. MXNet

29 Programming Interface

30 Example: SVM in MXNet

31 Example: SVM in MXNet (Cont)

32 Example: SVM in MXNet (Cont)



# Programming Interface

- ❶ Support many scope applications (e.g. computer vision, natural language processing, speech recognition, unsupervised machine learning, support embedded APIs, visualization)
- ❷ 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
- ❸ Fully compatible with Torch: modules and operators
- ❹ Support building neural network graphs
  - Call `mx.viz.plot_network()`
- ❺ Not well documented

# Example: SVM in MXNet

```
1 from __future__ import print_function
2 import mxnet as mx
3 import numpy as np
4 from sklearn.datasets import fetch_mldata
5 from sklearn.decomposition import PCA
6 # import matplotlib.pyplot as plt
7 import logging
8
9 logger = logging.getLogger()
10 logger.setLevel(logging.DEBUG)
11
12 # Network declaration as symbols. The following pattern was based
13 # on the article , but feel free to play with the number of nodes
14 # and with the activation function
15 data = mx.symbol.Variable('data')
16 fc1 = mx.symbol.FullyConnected(data = data, name='fc1', num_hidden=512)
17 act1 = mx.symbol.Activation(data = fc1, name='relu1', act_type="relu")
18 fc2 = mx.symbol.FullyConnected(data = act1, name = 'fc2', num_hidden = 512)
19 act2 = mx.symbol.Activation(data = fc2, name='relu2', act_type="relu")
20 fc3 = mx.symbol.FullyConnected(data = act2, name='fc3', num_hidden=10)
21
22 # Here we add the ultimate layer based on L2-SVM objective
23 mlp = mx.symbol.SVMOutput(data=fc3, name='svm')
24
25 # To use L1-SVM objective , comment the line above and uncomment the line below
26 # mlp = mx.symbol.SVMOutput(data=fc3, name='svm', use_linear=True)
27
28 # Now we fetch MNIST dataset , add some noise , as the article suggests ,
29 # permutate and assign the examples to be used on our network
30 mnist = fetch_mldata('MNIST original')
```

## Example: SVM in MXNet (Cont)

```
31 mnist_pca = PCA(n_components=70).fit_transform(mnist.data)
32 noise = np.random.normal(size=mnist_pca.shape)
33 mnist_pca += noise
34
35 # set seed for deterministic ordering
36 np.random.seed(1234)
37 p = np.random.permutation(mnist_pca.shape[0])
38 X = mnist_pca[p]
39 Y = mnist.target[p]
40 X_show = mnist.data[p]
41
42 # This is just to normalize the input to a value inside [0,1],
43 # and separate train set and test set
44 X = X.astype(np.float32)/255
45 X_train = X[:60000]
46 X_test = X[60000:]
47 X_show = X_show[60000:]
48 Y_train = Y[:60000]
49 Y_test = Y[60000:]
50
51 # Article's suggestion on batch size
52 batch_size = 200
53 train_iter = mx.io.NDArrayIter(X_train, Y_train, batch_size=batch_size)
54 test_iter = mx.io.NDArrayIter(X_test, Y_test, batch_size=batch_size)
55
56 # A quick work around to prevent mxnet complaining the lack of a softmax_label
57 train_iter.label = mx.io._init_data(Y_train, allow_empty=True, default_name='svm_label')
58 test_iter.label = mx.io._init_data(Y_test, allow_empty=True, default_name='svm_label')
```

## Example: SVM in MXNet (Cont)

```
60 # Here we instantiate and fit the model for our data
61 # The article actually suggests using 400 epochs,
62 # But I reduced to 10, for convinience
63 model = mx.model.FeedForward(
64     ctx = mx.cpu(0),          # Run on CPU 0
65     symbol = mlp,              # Use the network we just defined
66     num_epoch = 10,            # Train for 10 epochs
67     learning_rate = 0.1,       # Learning rate
68     momentum = 0.9,           # Momentum for SGD with momentum
69     wd = 0.00001,              # Weight decay for regularization
70 )
71
72 model.fit(
73     # Training data set
74     X=train_iter,
75     # Testing data set. MXNet computes scores on test set every epoch
76     eval_data=test_iter,
77     # Logging module to print out progress
78     batch_end_callback = mx.callback.Speedometer(batch_size, 200)
79 )
80
81 # Uncomment to view an example
82 # plt.imshow((X_show[0].reshape((28,28))*255).astype(np.uint8), cmap='Greys_r')
83 # plt.show()
84 # print 'Result:', model.predict(X_test[0:1])[0].argmax()
85
86 # Now it prints how good did the network did for this configuration
87 print('Accuracy:', model.score(test_iter)*100, '%')
```

## §6. Torch

33 Programming Interface

34 Example: Two-Layer Network

35 Example: Two-Layer Network (Cont)

36 Example: Linear Regression in Lua

37 Example: Linear Regression in Lua (Cont)

38 Example: Linear Regression in Lua (Cont)





# 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

```
1 import torch
2 from torch.autograd import Variable
3
4 # N is batch size; D_in is input dimension;
5 # H is hidden dimension; D_out is output dimension.
6 N, D_in, H, D_out = 64, 1000, 100, 10
7
8 # Create random Tensors to hold inputs and outputs, and wrap them in Variables.
9 x = Variable(torch.randn(N, D_in))
10 y = Variable(torch.randn(N, D_out), requires_grad=False)
11
12 # Use the nn package to define our model as a sequence of layers.
13 model = torch.nn.Sequential( torch.nn.Linear(D_in, H),
14                               torch.nn.ReLU(),
15                               torch.nn.Linear(H, D_out) )
16
17 # The nn package also contains definitions of popular loss functions;
18 loss_fn = torch.nn.MSELoss(size_average=False)
19
20 learning_rate = 1e-4
21
22 for t in range(500):
23     # Forward pass: compute predicted y by passing x to the model.
24     y_pred = model(x)
25     # Compute and print loss.
26     loss = loss_fn(y_pred, y)
27     print(t, loss.data[0])
28     # Zero the gradients before running the backward pass
29     model.zero_grad()
```

## Example: Two-Layer Network (Cont)

```
30     # Backward pass: compute gradient of the loss
31     loss.backward()
32
33     # Update the weights using gradient descent
34     for param in model.parameters():
35         param.data -= learning_rate * param.grad.data
36     end
37 end
```

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

# Example: Linear Regression in Lua

```
1  require 'torch'
2  require 'optim'
3  require 'nn'
4
5  # write the loss to a text file and read from there to plot it as training proceeds
6  logger = optim.Logger('loss_log.txt')
7
8  # input data
9  data = torch.Tensor{{40, 6, 4},{44, 10, 4},{46, 12, 5},
10 {48, 14, 7},{52, 16, 9},{58, 18, 12},{60, 22, 14},
11 {68, 24, 20},{74, 26, 21},{80, 32, 24}}
12
13 # define the container
14 model = nn.Sequential()
15 ninputs = 2; noutputs = 1
16
17 # define the only module
18 model:add(nn.Linear(ninputs , noutputs))
19
20 # Define a loss function
21 criterion = nn.MSECriterion()
22
23 # retrieve its trainable parameters
24 x, dl_dx = model:getParameters()
25
26 # compute loss function and its gradient
27 feval = function(x_new)
28   # set x to x_new, if different
29   if x ~= x_new then
30     x:copy(x_new)
31   end
```

## Example: Linear Regression in Lua (Cont)

```
32  # select a new training sample
33  _nidx_ = (_nidx_ or 0) + 1
34  if _nidx_ > (#data)[1] then _nidx_ = 1 end
35
36  local sample = data[_nidx_]
37  local target = sample[{ {1} }]
38  local inputs = sample[{ {2,3} }]
39
40  # reset gradients
41  dl_dx:zero()
42
43  # evaluate the loss function and its derivative wrt x
44  local loss_x = criterion:forward(model:forward(inputs), target)
45  model:backward(inputs, criterion:backward(model.output, target))
46
47  # return loss(x) and dloss/dx
48  return loss_x, dl_dx
49 end
50
51 # define SGD
52 sgd_params = {
53   learningRate = 1e-3,
54   learningRateDecay = 1e-4,
55   weightDecay = 0,
56   momentum = 0
57 }
58
59 # we cycle 10,000 times over our training data
60 for i = 1,1e4 do
61   #this variable is used to estimate the average loss
62   current_loss = 0
```

## Example: Linear Regression in Lua (Cont)

```
63  #an epoch is a full loop over our training data
64  for i = 1,(#data)[1] do
65      # return new x and value of the loss functions
66      _,fs = optim.sgd(feval,x,sgd_params)
67      # update loss
68      current_loss = current_loss + fs[1]
69  end
70
71  # report average error on epoch
72  current_loss = current_loss / (#data)[1]
73  print('current loss = ' .. current_loss)
74
75  logger:add[['training error'] = current_loss]
76  logger:style[['training error'] = '-']
77  logger:plot()
78 end
79
80 # Test the trained model
81 text = {40.32, 42.92, 45.33, 48.85, 52.37, 57, 61.82, 69.78, 72.19, 79.42}
82
83 for i = 1,(#data)[1] do
84     local myPrediction = model:forward(data[i][{{2,3}}])
85     print(string.format("%2d %6.2f %6.2f", i, myPrediction[1], text[i]))
86 end
```

## §7. Caffe

- 39 Programming Interface
- 40 Example: Image Classification
- 41 Example: Extend Layers
- 42 Example: Extend Layers (Cont)

Caffe

Deep learning framework  
by BAIR

# Programming Interface



- 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



# Example: Image Classification

```
1 import caffe
2 import matplotlib.pyplot as plt
3
4 # paste your image URL here
5 my_image_url = "https://wikipedia/Orang_Utan/2 C_Malaysia.JPG"
6 !wget -O image.jpg $my_image_url
7
8 # transform it and copy it into the net
9 image = caffe.io.load_image('image.jpg')
10 caffe.net.blobs['data'].data[...] = transformer.preprocess('data', image)
11
12 # perform classification
13 caffe.net.forward()
14
15 # obtain the output probabilities
16 output_prob = net.blobs['prob'].data[0]
17
18 # sort top five predictions from softmax output
19 top_inds = output_prob.argsort()[::-1][:5]
20
21 #
22 plt.imshow(image)
23
24 #
25 print 'probabilities and labels:'
26 zip(output_prob[top_inds], labels[top_inds])
```

# Example: Extend Layers

```
1 import caffe
2 import numpy as np
3
4 class EuclideanLoss(caffe.Layer):
5     def setup(self, bottom, top):
6         #check input pair
7         if len(bottom) != 2:
8             raise Exception("Need two inputs to compute distance")
9
10    def reshape(self, bottom, top):
11        #check input dimensions match
12        if bottom[0].count != bottom[1].count:
13            raise Exception("Inputs must have the same dimension")
14        #difference in shape of inputs
15        self.diff = np.zeros_like(bottom[0].data, dtype=np.float32)
16        # loss output is scalar
17        top[0].reshape(1)
18
19    def forward(self, bottom, top):
20        self.diff[...] = bottom[0].data - bottom[1].data
21        top[0].data[...] = np.sum(self.diff**2)/bottom[0].num/2.
22
23    def backward(self, top, propagate_down, bottom):
24        for i in range(2):
25            if not propagate_down[i]:
26                continue
27            if i == 0:
28                sign = 1
29            else:
30                sign = -1
```

## Example: Extend Layers (Cont)

```
31 bottom[i].diff[...] = sign.self.diff / bottom[1].num
```

### Define a class in Python to extend Layer

```
32 layer{  
33     type: "Python"  
34     python_param {  
35         module: "layers"  
36         layer: "EuclideanLoss"  
37     }  
38 }
```

Thank You!

