Software Packages for Deep Learning

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

Outline

§1. Introduction

Machine Learning

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

General Tasks of ML

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

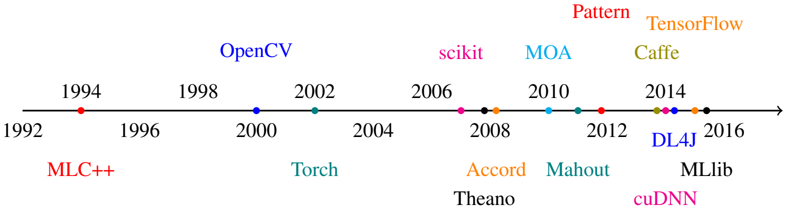
Packages for General Machine Learning

What is the purpose?

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

What do we want for a ML package?

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



A Tour of TensorFlow, by Peter Goldsborough.

Deep Learning: Pros and Cons

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

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

Pros:

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

Cons:

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

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

Popular Packages: Basic Information

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

- Other worth noting packages: CNTK/DMTK (Microsoft), PyTorch (beta)
- BVLC, Berkeley Vision and Learning Center
- DMLC, Distributed (Deep) Machine Learning Community, supported by Amazon, Intel, Microsoft, nVidia, Baidu, ...

Popular Packages: Performance

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

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

§2. Numerical Comparisons

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 setting for data parallelization

Benchmarking State-of-the-Art Deep Learning Software Tools, by Shaohuai Shi, Qiang Wang.

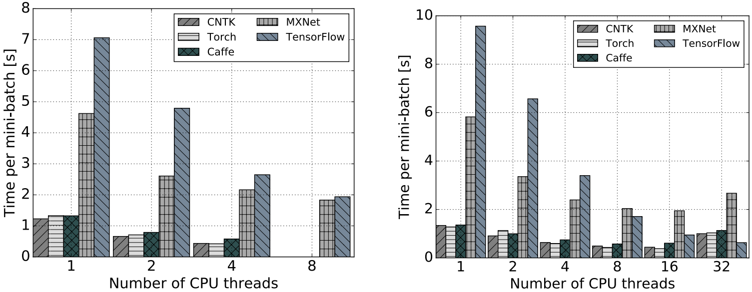
Neural Networks and Data Sets

- 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
CNN	AlexNet-S	150528	1000	4	~61 millions
FCN	FCN-R	784	10	5	~31 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

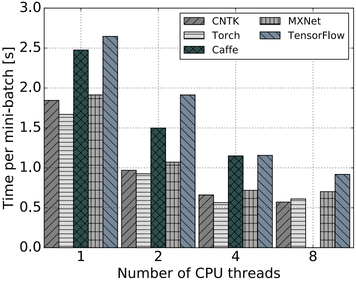
CPU Scalability: FCN Synthetic



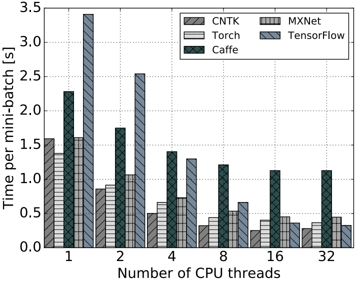
(a) Results on i7-3820. (b) Results on E5-2630.

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

CPU Scalability: FCN Real



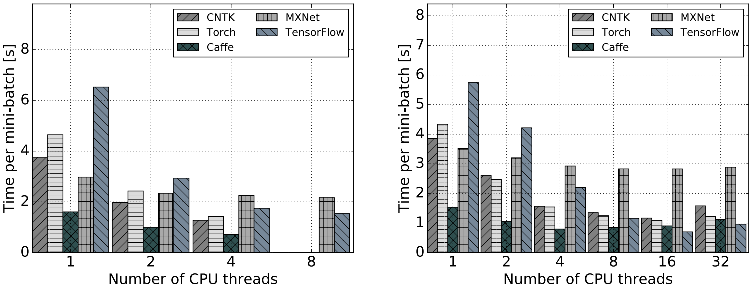
(a) Results on i7-3820.



(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.)

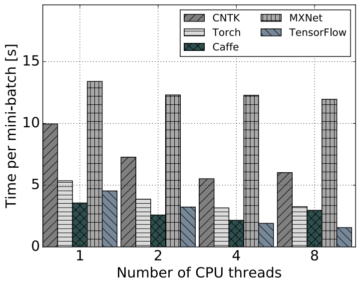
CPU Scalability: CNN Synthetic



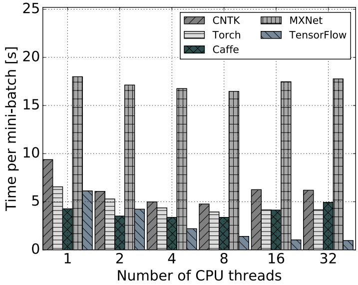
(a) Results on i7-3820. (b) Results on E5-2630.

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

CPU Scalability: CNN Real



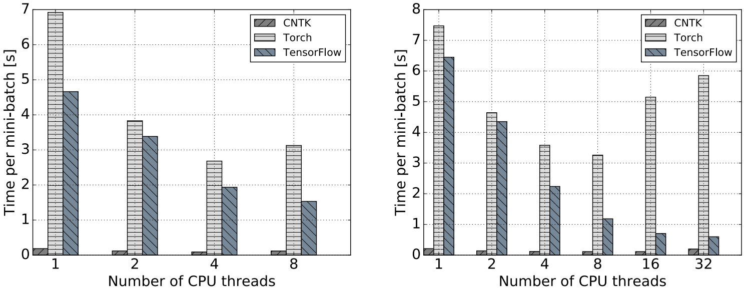
(a) Results on i7-3820.



(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



(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

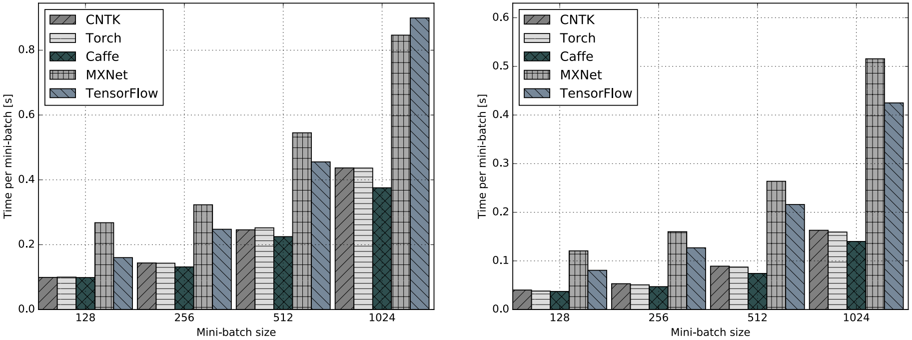


Figure: The performance comparison of FCN-S on GPU platforms.

GPU Scalability: CNN Synthetic

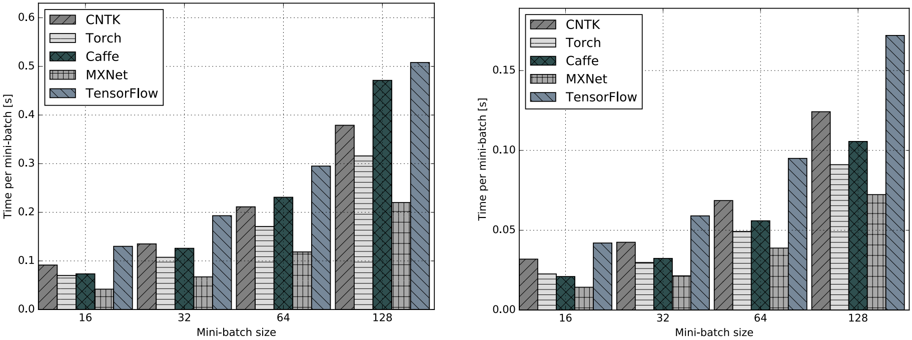


Figure: The performance comparison of AlexNet-S on GPU platforms.

GPU Scalability: CNN Real

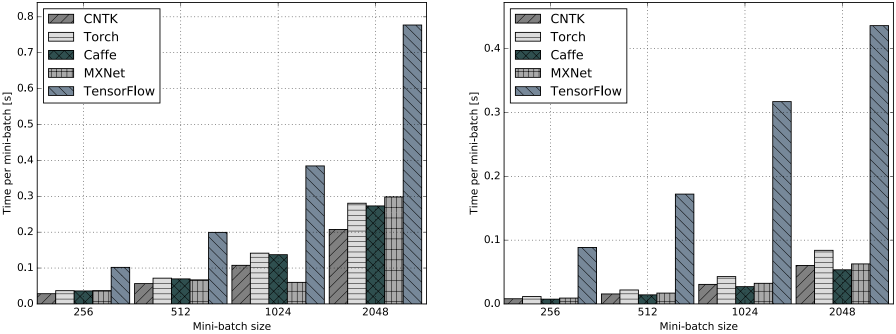
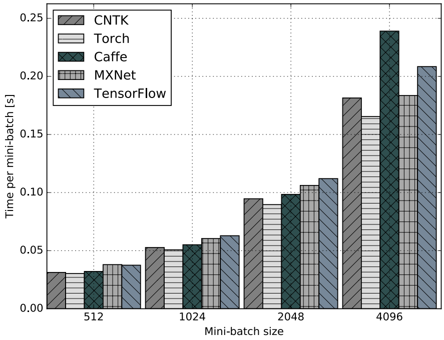


Figure: The performance comparison of AlexNet-R on GPU platforms.

GPU Scalability: RNN Real



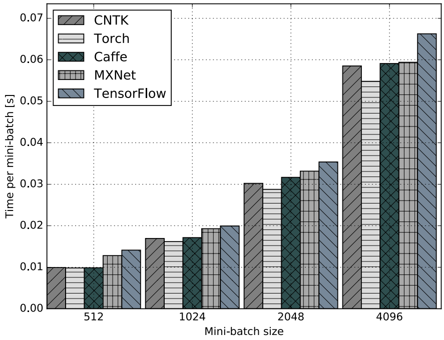
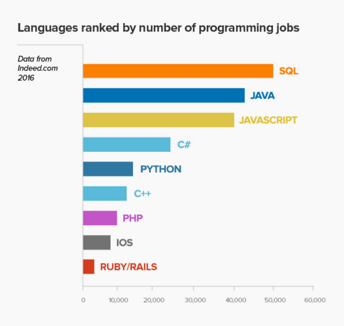


Figure: The performance comparison of FCN-R on GPU platforms.

§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().

Modules

Using modules

1. 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. For example: math.sin(3.14).

```
import math
```

2. This way does not introduce the name math into the current namespace. It does however introduce all public names of the math module into the current namespace, directly using: sin(3.14)

```
from math import *
```

3. This will only import the sin function from math module and introduce the name sin into the current namespace, but it will not introduce the name math into the current namespace, directly using: sin(3.14)

```
from math import sin
```

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

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

Control Flow

If-then-else

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

For loop

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

While loop

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

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. When objects are passed to a function, Python always passes (the value of) the reference to the object to the function.

3. help() is a function which gives information about the object. For example, help('modules') will generate a list of all modules which can be imported into the current interpreter.

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. Torch

Programming interface

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

Example: Linear Regression

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

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

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

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

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

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

Example: Linear Regression

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

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

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

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

# reset gradients
   dl_dx:zero()
```

Example: Linear Regression

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

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

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

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

Example: Linear Regression

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

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

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

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

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

Example: 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)
```

Example: Two-Layer Network

```
learning_rate = 1e-4

for t in range(500):
    # Forward pass: compute predicted y by passing x to the model.
    y_pred = model(x)

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

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

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

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

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

§5. Caffe

Programming interface

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

Example: 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.
```

Example: Extend Layers

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

Define a class in Python to extend Layer

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

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

§6. TensorFlow

Computational graph

```python
import tensorflow as tf

graph = tf.Graph()

with graph.as_default():
    a = tf.constant(1.0)
    tf.summary.scalar('aa', a)
    b = tf.constant(2.0)
    tf.summary.scalar('bb', b)
    c = tf.multiply(a, b)
    tf.summary.scalar('c', c)

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

with tf.Session(graph=graph):
    tf.global_variables_initializer().run()
    writer.add_summary(merged.eval())
```

Figure: Computaion graph

Programming interface

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

Visualization: TensorBoard

Computation graphs are powerful but complicated

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

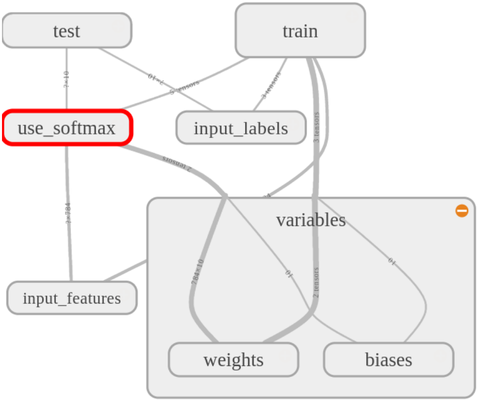


Figure: Graph Visualization

Example 1: SoftMax

```
import tensorflow as tf

# Import 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 graph
graph = tf.Graph()
with graph.as_default():
    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')
        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 labels y
        y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Example 1:SoftMax

```
    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, minimizing the entropy, 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"))
        tf.summary.scalar('Accuracy', accuracy)
    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter('./board', graph)

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 = {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)
```

§7. MxNET

Programming interface

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

Example 1: SoftMax

```python

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

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

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

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

Example 1: SoftMax

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

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

# Calculate gradient function automatically
grad_function = grad_and_loss(train_loss)

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


Thank You!