Software Packages for Deep Learning

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

Outline

Machine Learning

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

General Tasks of ML

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

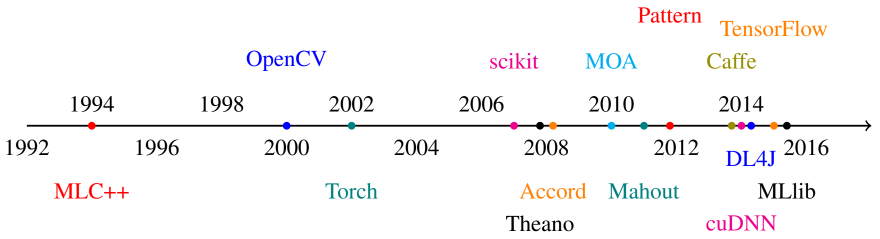
Packages for General Machine Learning

What is the purpose?

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

What do we want for a ML package?

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



Deep Learning: Pros and Cons

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

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

Pros:

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

Cons:

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

Comparison: Basic Information

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

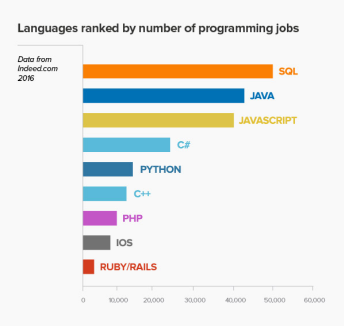
- BVLC, Berkeley Vision and Learning Center
- DMLC, Distributed (Deep) Machine Learning Community, supported by Amazon, Intel, Microsoft, nVidia, Baidu, ...

Comparison: Performance

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

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().

Functions and Modules

Defining functions

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

Using modules

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

Built-in Data Structures

Numeric types: int, float, complex

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

Sequence types: list, tuple, str, dict

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

Formatted output

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

Control Flow

If-then-else

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

For loop

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

While loop

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

http://www.pythontutor.com

Programming interface

Example 1

Programming interface

Example 1

Computational graph

TensorFlow computations are expressed as stateful dataflow graphs.

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

```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0)
node3 = tf.add(node1, node2)
add_and_triple = adder_node * 3
```

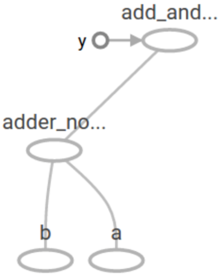


Figure: Computaion graph

Programming interface

Visualization: TensorBoard

Computation graphs are powerful but complicated

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

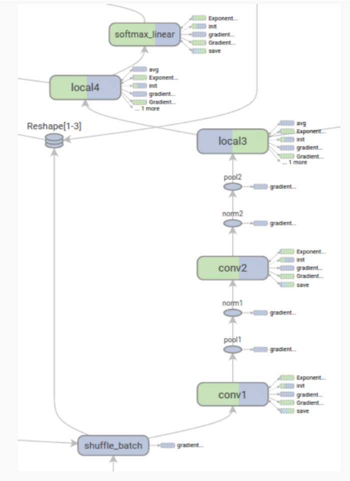


Figure: Graph Visualization

Example 1: SoftMax

```
import tensorflow as tf

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

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

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

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

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

Example 1:SoftMax

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

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

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

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

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

Programming interface

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)
```

Numerical tests


Thank You!