Software Packages for Deep Learning

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

Outline

Machine Learning

- ML gives computers the ability to learn without being explicitly programmed [Samuel 1959]
- ML explores the study and construction of algorithms that can learn from and make predictions on data
- Data mining, computational statistics, optimization, ...
- Fourth paradigm, big data, deep learning, 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 (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
- Other tasks: density estimation, dimensionality reduction, ...

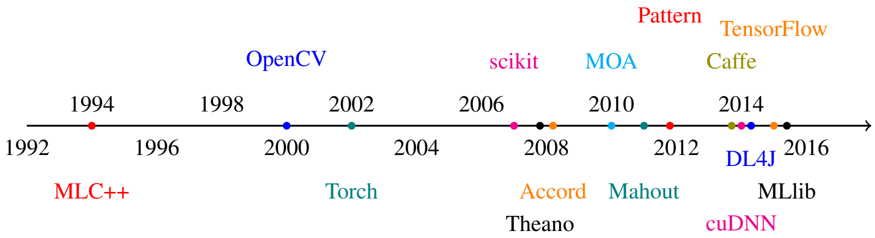
Packages for General Machine Learning

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

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-calss at pattern recongition

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

Table: Framework 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++ Python
Supported Interface	Lua	C++/Python Matlab	C++/Python R/Java/Go	C++/Python R/Julia/Scala

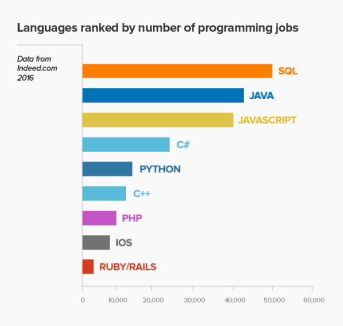
Comparison

Table: Framework Comparision: Performance

Viewpoint	Torch	Caffe	TensorFlow	MXNet
Pretrained Models	Yes	Yes	No	Yes
Low-level Operators	Good	Good	Fairly good	Very few
High-level Support	Good	Good	Good	Good
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?

- Strong introspection[1] capabilities (???What does even mean???)
- Full modularity, supporting hierarchical packages
- Exception-based error handling
- Dynamic data types and automatic memory management

Why consider such a slow language for simulation?

- Good for proof-of-concept
- 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

Built-in Data Structures

- Numeric types–int, float, complex

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

- Sequence types–list, tuple, str, dict

```
For example:
g=[3.14, True, 'Yes', [1], (1L,)] + [None]*3, list
h=(3.14, True, 'Yes', [1], ()), tuple
i='Hello' + ", " + '''world!''', str
j=\{1: 'int', 'pi': 3.14\}, dict
```

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
```

Functions and Modules

- Defining functions

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

- Using modules

  There are 3 different ways to use modules. Examples are below.

  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)

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
```

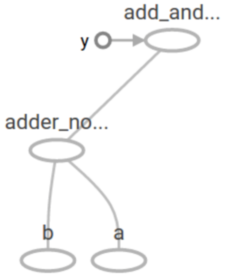


Figure: Computaion graph

Programming interface

Visualization: TensorBoard

Computation graphs are powerful but complicated

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

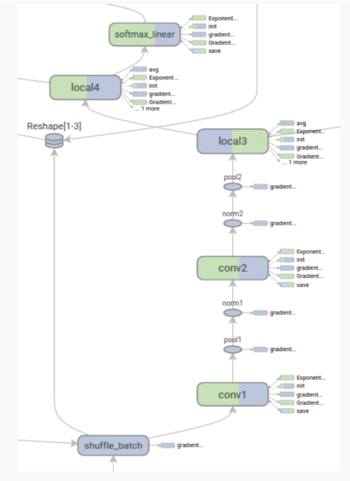


Figure: Graph Visualization

Example 1: SoftMax

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()

# model
Y= tf.nn.softmax(tf.matmul(tf.reshape(X,[-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

Example 1:SoftMax

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)

sess = tf.Session()
sess.run(init)

for i in range(10000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
        train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ? add code to print it
    a,c = sess.run([accuracy, cross_entropy], feed=train_data)

    # success on test data ?
    test_data={X:mnist.test.images, Y_:mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

Programming interface

Example 1

Numerical tests


Thank You!