TensorFlow

Introduction to TensorFlow

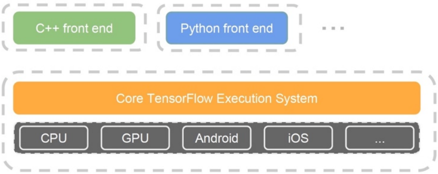
Wenjian Huang
huangwenjian@ppmoney.com

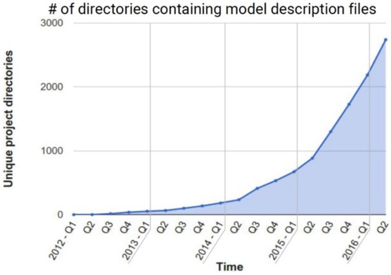
July 13, 2017

What is TensorFlow

TensorFlow [1] is an interface for expressing machine learning (not only deep learning) algorithms, and an implementation for executing such algorithms.

code: https://github.com/tensorflow/tensorflow
models: https://github.com/tensorflow/models





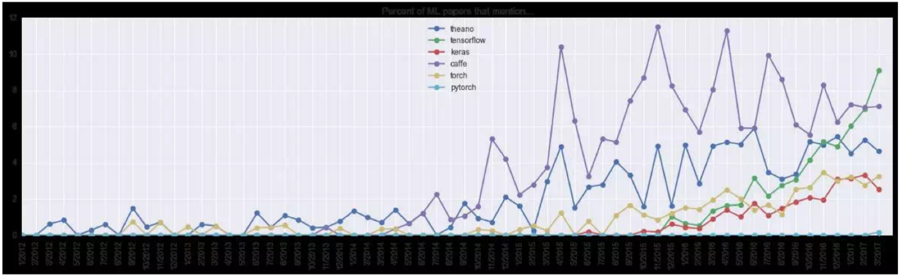
[1] www.tensorflow.com

Trending in TensorFlow and other Deep Learning Frameworks

Released in 2015 November

Top 1 Machine Learning / Deep Learning / Python Project in GitHub

More users than other frameworks and better documentations



Comparison with other Frameworks

Frameworks	Organizations	Supported Languages	Stars	Forks	Contributors
TensorFlow	Google	Python/C++/Go/..	62660	30339	936
Caffe	BVLC	C++/Python	18870	11587	247
Keras	fchollet	Python	17278	6169	484
CNTK	Microsoft	C++/Python	11662	2953	135
MXNet	DMLC	Python/C++/R/...	10280	3848	380
Torch7	Facebook	Lua	7060	2098	129
Deeplearning4J	DeepLearning4J	Java/Scala	6889	3299	119
Theano	U. Montreal	Python	6584	2191	310
PyTorch	Facebook	Python	5720	1072	248
Caffe2	Facebook	C++/Python	5127	1056	93
PaddlePaddle	Baidu	C++/Python	5072	1342	71
Sonnet	DeepMind	Python	5040	598	11
Neon	NervanaSystems	Python	3092	675	62

updated on July 6, 2017

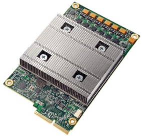
Start Up

The Google Brain project started in 2011 to explore the use of very-large-scale deep neural networks :

- **DistBelief** : the first generation scalable distributed training and inference system
- **TensorFlow** : second-generation system for the implementation and deployment of large scale machine learning models

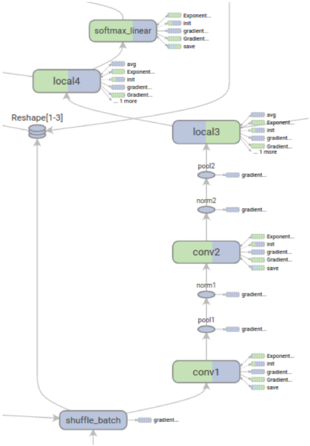
TensorFlow Research Cloud (1000 TPUs, free for top researcher)

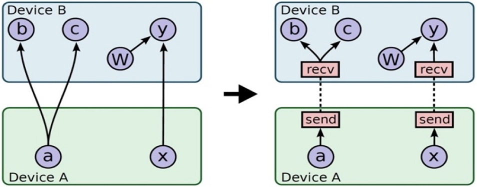
https://www.tensorflow.org/tfrc/



Data Flow Model

Data flow graphs describe mathematical computation with a directed acyclic graph (DAG) of nodes & edges, allowing some kinds of nodes to maintain and update persistent state and for branching and looping control structures.





Programming Model

Basic Concepts

- **Operation** : An operation has a name and represents an abstract computation. (e.g., "add" and "dot product")
- **Kernel** : An implementation of an operation that can run on a particular type of device. (e.g., "CPU" and "GPU")
- **Session** : Sessions manage and execute TensorFlow computation graphs. Therefore, client programs interact with the TensorFlow system through sessions.
- **Tensor** : A n-dimensional array that flows along the edges of the computation graph.
- **Variable** : A special kind of operation that returns a handle to a persistent mutable tensor that survives across executions of a graph.

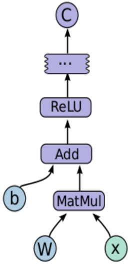
Programming Steps

- Represents computations as graphs.
- Represents data as tensors.
- Maintains state with Variables. Uses feeds and fetches to get data into and out of arbitrary operations.
- Executes graphs in the context of Sessions.

Code Example

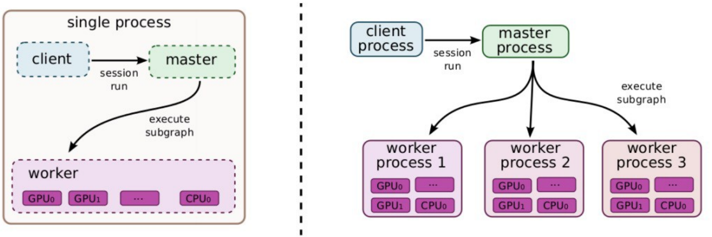
```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                   # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                       # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)             # Relu(Wx+b)
C = [...]                                          # Cost computed as a function
                                                   # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...       # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})          # Fetch cost, feeding x=input
  print step, result
```

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration



Underlying Implementations

The main components in a TensorFlow system are the *client*, which uses the Session interface to communicate with the master and one or more worker processes, with each worker process responsible for arbitrating access to one or more computational devices (such as CPU cores and GPU cards).

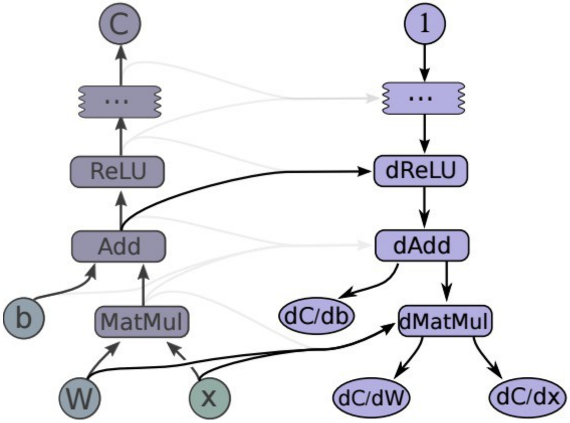


Extensions

- **Automatic Differentiation** : Automatically computes gradients for data flow graphs.
- **Partial Execution** : Allows TensorFlow clients to execute a subgraph of the entire execution graph.
- **Device Constraints** , **Control Flow**, **Input Operations** , **Queues** , **Containers** $\cdots$

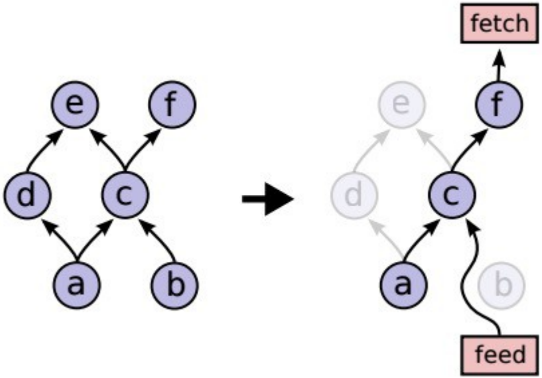
Gradient Computation

TensorFlow has built-in support for automatic gradient computation



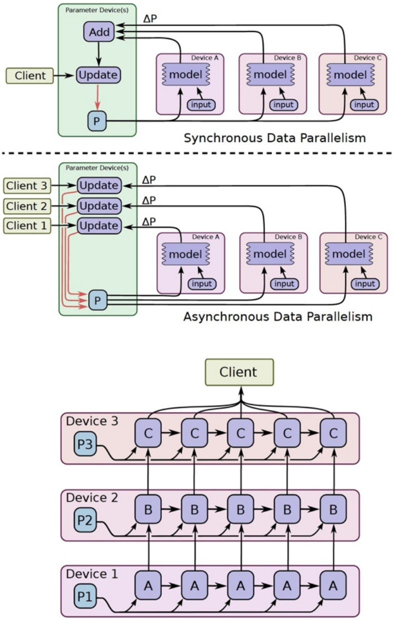
Partial Execution

To execute an arbitrary subgraph of the whole graph

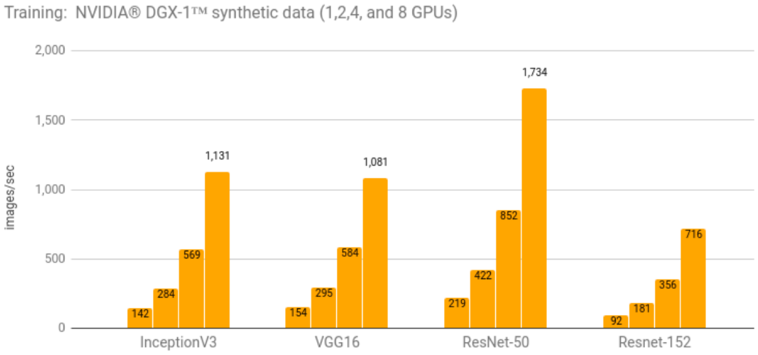


Features Summary

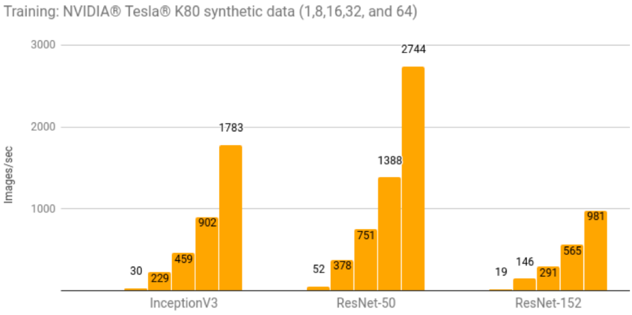
Programming Model	• Dataflow-like model
Language	• Python • C++
Deployment	• Code once, Run everywhere
Computing Resource	• CPU • GPU
Distribution Process	• Local Implementation • Distributed Implementation
Math Expressions	• Math Graph Expression • Auto Differentiation
Optimization	• Auto Elimination • Kernel Optimization • Communication Optimization • Support model, data parallelism • ...

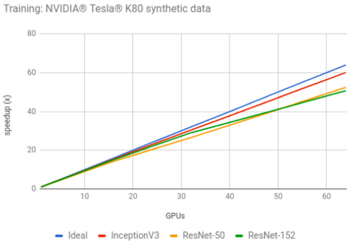


Multi-GPU in Single Machine

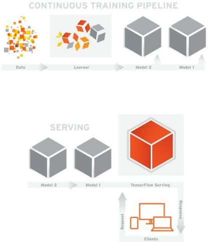


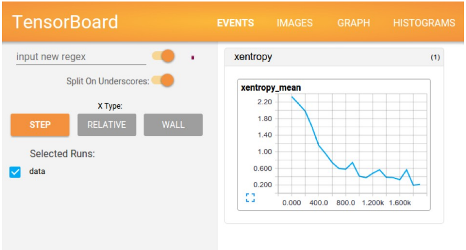
Distributed Training

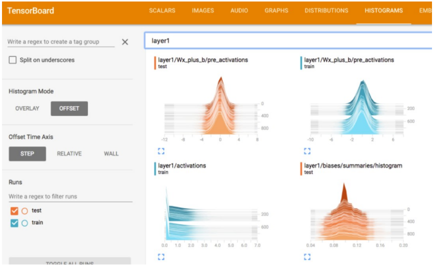


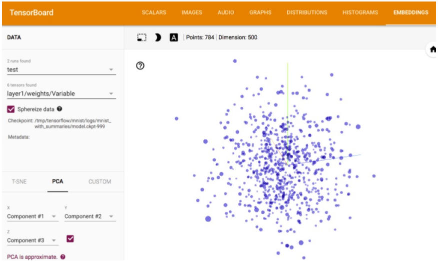


TensorFlow Serving & TensorBoard









Native API on FCN,CNN,RNN

FCN

```
W1 = tf.Variable(tf.truncated_normal([in_units, h1_units], stddev=0.1))
b1 = tf.Variable(tf.zeros([h1_units]))
W2 = tf.Variable(tf.zeros([h1_units, 10]))
b2 = tf.Variable(tf.zeros([10]))
hidden1 = tf.nn.relu(tf.matmul(x, W1) + b1)
y = tf.nn.softmax(tf.matmul(hidden1, W2) + b2)
```

CNN

```
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 32], stddev=0.1))
b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]))
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                          padding='SAME')
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

Native API on FCN,CNN,RNN

Embedding & LSTM

```
word_vectors = tf.contrib.layers.embed_sequence(features, vocab_size=n_words,
                embed_dim=EMBEDDING_SIZE, scope='words')
word_list = tf.unstack(word_vectors, axis=1)

cell = tf.nn.rnn_cell.BasicLSTMCell(EMBEDDING_SIZE)
outputs, state = tf.nn.static_rnn(cell, inputs=word_list, dtype=tf.float32)
```

Loss & Optimizer

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
                               reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

tf.contrib.layers(slim) API on FCN,CNN

Automatically set default activation_fn, weights_initializer, bias_initializer

activation_fn → tf.nn.relu
weights_initializer → initializers.xavier_initializer()
bias_initializer → tf.zeros_initializer()

```
layers = tf.contrib.layers # tf.contrib.slim
```

FCN

```
hidden1 = layers.fully_connected(x, h1_units)
y = layers.fully_connected(x,10,activation_fn=layers.softmax)
```

CNN

```
h_conv1 = layers.conv2d(inputs,num_outputs=32,kernel_size=[8,8],stride=[4,4],
              padding='VALID')
h_pool1 = layers.max_pool2d(h_conv1, [2,2])
```

tf.contrib.keras API on FCN,CNN,RNN

```
keras = tf.contrib.keras
model = keras.models.Sequential()
```

FCN

```
model.add(keras.layers.Dense(h1_units,input_dim=in_units,activation='relu'))
model.add(keras.layers.Dense(10,activation=’softmax’)
```

CNN

```
model.add(keras.layers.Conv2D(32,[5,5],input_shape=[28,28,3]))
model.add(keras.layers.MaxPool2D())
```

RNN

```
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
```

Loss & Optimizer

No need for explicit loss define, done by ’compile’ the model

```
model.compile(loss='categorical_crossentropy', optimizer=SGD(0.1))
model.compile(loss='mse', optimizer=Adagrad(lr=0.01,epsilon=1e-3))
```

TIPS

GPU Visibility: Sometimes you want the program to use a specific GPU

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="1"
```

Memory Growth: Allow GPU memory growth instead of using up all memory

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

Reset the graph and re-run the training

```
tf.reset_default_graph()
```

TIPS

Make your result reproducible

```
import numpy as np
np.random.seed(0)
import tensorflow as tf
tf.set_random_seed(0)
```

Accelerate MLP

```
config = tf.ConfigProto()
config.graph_options.optimizer_options.global_jit_level =
tf.OptimizerOptions.ON_1
sess = tf.InteractiveSession(config=config)
```

TIPS

Utilize queues instead of feed_dict, or preload all data into GPU memory using Variable when data is small

```
queue = tf.train.string_input_producer(["file0.csv","file1.csv"])
k, v = tf.TextLineReader().read(queue)
cols = tf.decode_csv(v)
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess=sess, coord=coord)
```

Preprocessing on CPU not GPU (save communication time, let GPU focus on training)

```
with tf.device('/cpu:0'):
    preprocessing...
```

Fused Batch Norm and NCHW format, NCHW train faster on GPU through cuDNN, NHWC run faster on CPU during inference

```
bn = tf.contrib.layers.batch_norm(
         input_layer, fused=True, data_format='NCHW'
         scope=scope, **kwargs)
```