Deep Learning Frameworks

COSC 7336: Advanced Natural Language Processing
Fall 2017
Today’s lecture

★ Deep learning software overview
★ TensorFlow
★ Keras
★ Practical
Graphical Processing Unit (GPU)

- From graphical computing to general numerical processing GPGPU
- Single Instruction Multiple Data architecture
- High-throughput type computations with data-parallelism
- Commodity hardware
- Two main vendors: NVidia, AMD
Performance evolution
# NVIDIA Tesla

## PERFORMANCE SPECIFICATIONS FOR NVIDIA TESLA P4, P40 AND P100 ACCELERATORS

<table>
<thead>
<tr>
<th></th>
<th>Tesla P4 for Ultra-Efficient Scale-Out Servers</th>
<th>Tesla P40 for Maximum-Inference Throughput Servers</th>
<th>Tesla P100: The Universal Datacenter GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Precision Performance (FP32)</td>
<td>5.5 TeraFLOPS</td>
<td>12 TeraFLOPS</td>
<td>10.6 TeraFLOPS</td>
</tr>
<tr>
<td>Half-Precision Performance (FP16)</td>
<td>--</td>
<td>--</td>
<td>21 TeraFLOPS</td>
</tr>
<tr>
<td>Integer Operations (INT8)</td>
<td>22 TOPS*</td>
<td>47 TOPS*</td>
<td>--</td>
</tr>
<tr>
<td>GPU Memory</td>
<td>8 GB</td>
<td>24 GB</td>
<td>16 GB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>192 GB/s</td>
<td>346 GB/s</td>
<td>732 GB/s</td>
</tr>
<tr>
<td>System Interface</td>
<td>Low-Profile PCI Express Form Factor</td>
<td>Dual-Slot, Full-Height PCI Express Form Factor</td>
<td>Dual-Slot, Full-Height PCI Express Form Factor, or SXM2 Form Factor with NVLink</td>
</tr>
<tr>
<td>Power</td>
<td>50 W/75 W</td>
<td>250 W</td>
<td>250 W (PCIe) 300W (SXM2)</td>
</tr>
<tr>
<td>Hardware-Accelerated Video Engine</td>
<td>1x Decode Engine, 2x Encode Engines</td>
<td>1x Decode Engine, 2x Encode Engines</td>
<td>--</td>
</tr>
</tbody>
</table>
GPU vs CPU for deep learning
Programming GPUs

★ CUDA:
  ○ NVidia’s parallel computing platform
  ○ Access to the GPU's virtual instruction for the execution of compute kernels on the parallel computational elements.
  ○ CUDA C: a specialized version of C (also CUDA Fortran)
  ○ Optimized libraries

★ OpenCL:
  ○ Similar to CUDA but multiplatform no vendor dependant.
  ○ The way to go with AMD GPU cards.
  ○ A step behind CUDA
Deep learning frameworks

Caffe2
PYTORCH
MXNet
Facebook
TensorFlow
Theano
Keras
My Advice:

**TensorFlow** is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (Keras, Sonnet, etc)

I think **PyTorch** is best for research. However still new, there can be rough patches.

Use **TensorFlow** for one graph over many machines

Consider **Caffe, Caffe2, or TensorFlow** for production deployment

Consider **TensorFlow or Caffe2** for mobile
Overview

★ Numerical computation based on dataflow graphs
★ Developed in C++
★ Python and C++ frontends
★ Automatic differentiation
★ Easy visualization using TensorBoard
★ Abstraction layers
  ○ Tf.contrib.learn, tf.contrib.slim
  ○ TFLearn, Keras
★ Support of heterogeneous architectures: multi-CPU, GPU, multi-GPU, distributed, mobile
Computation graphs (CG)

★ A CG defines the operations that have to be performed over a set of constants and variables.
★ TF works over CG where the variables are usually tensors (scalars, vectors, matrices, multidimensional matrices).
★ In TF the CG is first created and then it can be executed.
★ CG can be symbolically manipulated: e.g. to calculate its gradient or to simplify it.

\[ ax^2 + bx + c \]
Creating a graph in TF

```python
import tensorflow as tf

graph = tf.Graph()

with graph.as_default():
    a = tf.constant(10, tf.float32, name='a')
    b = tf.constant(-5, tf.float32, name='b')
    c = tf.constant(4, tf.float32, name='c')

    x = tf.placeholder(tf.float32, name='x')

    y = a * x * x + b * x + c

show_graph(graph.as_graph_def())
```
Executing a graph

- Executing a graph requires to create a session.
- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
- Sessions have to be closed so that assigned resources are released.

```python
import tensorflow as tf

# Graph definition
a = tf.constant(10, tf.float32, name= 'a')
b = tf.constant(-5, tf.float32, name= 'b')
c = tf.constant(4, tf.float32, name= 'c')
x = tf.placeholder(tf.float32, name= 'x')
y = a * x * x + b * x + c

# Graph execution
sess = tf.Session()
result = sess.run(y, {x: 5.0})
sess.close()
print(result)
```
Tensors

★ In general, a tensor is a multidimensional array:
  ○ Vector: one dimensional tensor.
  ○ Matrix: two dimensional tensor.

★ In TF, a tensor is a symbolic handle to one of the outputs of an operation.
★ It does not hold the values of that operation’s output, but instead provides a means of computing those values in a session.
★ The two main attributes of a tensor are its data type and its shape.

```python
import tensorflow as tf

a = tf.constant(10)
b = tf.constant([1, 2.5, 3])
c = tf.constant([[[1, 2], [3, 4]],
                 [[11, 12], [13, 14]],
                 [[21, 22], [23, 24]]])

print(a)
print(b)
print(c)
```

Tensor("Const_24:0", shape=(), dtype=int32)
Tensor("Const_25:0", shape=(3,), dtype=float32)
Tensor("Const_26:0", shape=(3, 2, 2), dtype=int32)
A variable maintains state in the graph across calls to run()

The Variable() constructor requires an initial value for the variable, which can be a Tensor of any type and shape.

The initial value defines the type and shape of the variable.

Placeholders allow to input values to the graph.

Placeholder values must be fed using the feed_dict optional argument to Session.run().

```python
import tensorflow as tf

# Graph definition
a = tf.constant(10, tf.float32, name= 'a')
b = tf.constant(-5, tf.float32, name= 'b')
c = tf.constant(4, tf.float32, name= 'c')
x = tf.placeholder(tf.float32, name= 'x')
y = a * x * x + b * x + c

# Graph execution
with tf.Session() as sess:
    result = sess.run(y, {x: 5.0})
sess.close()

print(result)
print(1, val_x, val_y)
sess.close()
```
Optimization

★ TF can automatically calculate gradients of a graph.
★ You can use the gradients to implement your own optimization strategy,
★ or you can use optimization methods already implemented in the system.
★ Parameters to optimize must be declared as variables.
★ When an optimizer instance is created, it receives parameters such as the learning rate.
★ Optimizer must called with the objective function.
★ Variables must be initialized.

```python
import tensorflow as tf

# Graph definition
a = tf.constant(10, tf.float32, name='a')
b = tf.constant(-5, tf.float32, name='b')
c = tf.constant(4, tf.float32, name='c')
x = tf.Variable(0.0, name='x')
y = a * x * x + b * x + c

optimizer = tf.train.GradientDescentOptimizer(0.02)
update = optimizer.minimize(y)

# Graph execution
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(20):
    val_y, val_x, _ = sess.run([y, x, update])
    print(i, val_x, val_y)
sess.close()
```

0 0.1 4.0
1 0.16 3.6
2 0.196 3.456
3 0.2176 3.40416
4 0.23056 3.3855
5 0.238336 3.37878
6 0.243002 3.37636
7 0.245801 3.37549
Optimization CG
Optimization CG
Optimizers

- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer
Monitoring with TensorBoard

★ TensorBoard is a visualization application provided by TensorFlow.
★ It visualizes summary data which is written to log files during training.
★ It also visualizes the computing graph as well as complementary information such as images.
Devices

★ TF supports different target devices: CPU, GPU, multi GPU
★ A graph can be distributed among different devices
★ TF takes care of consolidating the data

```python
# Creates a graph.
c = []
for d in ['/gpu:2', '/gpu:3']:
    with tf.device(d):
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
        c.append(tf.matmul(a, b))
    with tf.device('/cpu:0'):
        sum = tf.add_n(c)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(sum))
```
Additional topics

- **Estimators**: a high-level TF API that greatly simplifies machine learning programming. Encapsulates main ML tasks: training, evaluation, prediction.

- **Saving and loading models**: TF provides different tools to persists trained models.

- **Dataset API**: makes it easy to deal with large amounts of data, different data formats, and complicated transformations.

- **tf.layers**: provides a high-level API that makes it easy to construct a neural network. It provides methods that facilitate the creation of dense (fully connected) layers and convolutional layers.

- **tf.nn**: Neural network support.

- **tf.contrib**: contains volatile or experimental code.
TensorFlow Demo
Keras

★ Developed by François Chollet
★ High-level Python framework able to run on top of TensorFlow, Theano or CNTK,
★ Guiding principles:
  ○ User friendliness
  ○ Modularity
  ○ Easy extensibility
  ○ Work with Python
★ Highly popular
★ Fast prototyping
★ Easy to extend
★ Many pretrained models
Sequential model

The simplest model is sequential.
Layers are stacked one above the other.
The learning process is configured with compile.
Training is performed with one line.
The trained model can be easily evaluated.
And applied to new data.

```python
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense, Activation
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```
The functional API

★ The sequential model is easy to use, but someway restricted.
★ The functional API gives more flexibility that allows to construct more complex models:
  ○ Multiple outputs (multi-task)
  ○ Multi inputs
  ○ Shared layers
Layers

★ Layers are the building blocks of models
★ Keras provides several predefined layers for building different types of networks
★ Layers have different methods that allow to get and set their weights, to define an initialization function, to control the regularization, the activation function etc.
Preprocessing

★ Sequences:
  ○ Pad_sequences
  ○ Skip-grams

★ Text
  ○ Text to word sequence
  ○ One hot
  ○ Hashing
  ○ Tokenizer

★ Images
  ○ Normalization
  ○ Data augmentation
Keras Demo