Overview of Deep Learning Stack

Wahid Bhimji, Prabhat, Mustafa Mustafa, Steve Farrell

New User Training
June 21 2019
Deep Learning Stack on HPC

Technologies

Deep Learning Frameworks
- TensorFlow
- PyTorch
- Caffe
- Neon, CNTK, MXNet, ...

Multi Node libraries
- Cray ML PE Plugin
- Horovod
- MLSL
- MPI
- GRPC

Single Node libraries
- MKL-DNN
- CuDNN

Hardware
- CPUs (KNL)
- GPUs
- FPGAs
- Accelerators
Software Frameworks

- Different frameworks popularity has evolved rapidly
- % of ML Papers that mention a framework (up to Mar 2018)

- Caffe and Theano most popular 3-4 years ago
- Then Google released TensorFlow which now dominates
- PyTorch is recently rising rapidly in popularity
- See also DL power scores (also rates Tensorflow top):
  https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Source: https://twitter.com/karpathy/status/972295865187512320?lang=en
Subjective evaluation

• **TensorFlow:**
  – Reasonably easy to use directly within python (not as easy as with Keras)
  – Very nice tools for development like TensorBoard
  – Active development for features (e.g. dynamic graphs) and performance (e.g. for CPU/KNL) and ease of use

• **Keras:**
  – High-level framework sits on top of tensorflow (or theano) (and now part of TensorFlow)
  – Very easy to create standard and even advanced deep networks with a lot of templates/ examples
Subjective evaluation

- **PyTorch**
  - Relatively recent python adaption of ‘torch’ framework - heavily contributed to by Facebook
  - More pythonic than Tensorflow/Keras
  - Dynamic graphs from the start - very flexible
    - Popular with some ML researchers
  - Previously some undocumented quirks but Version 1.0 release added stability and performance

- **Caffe**
  - Optimised performance (still best for certain NN on CPUs)
  - Relatively difficult to develop new architectures
  - Caffe2 and PyTorch projects merged
• **Easiest is to use default anaconda python:**
  module load python
  python
  >>>> import tensorflow as tf

• **Active work by Intel to optimize for CPU:**
  – Available in anaconda. Modules on Cori:
    module avail tensorflow  #Display versions
    module load tensorflow
  – Loads the default (**intel-1.13.1-py36** as of Jun 2019)

• **Can also tune variables for performance (e.g. see [intel blog](#)):**
  – E.g Inter-op and Intra-op
TensorBoard

• Easy, customisable, visualization of training in progress
• At NERSC run TensorBoard on login node; point to logs made by jobs on compute node (chose an unused port)
  
  cori05 > tensorboard --logdir=path/to/logs --port 9998

• Use a ssh tunnel from your laptop to connect then open localhost:9998 in your browser (note: others will also be able to see your experiments if they connect to that port)
  
  YourLaptop > ssh -L 9998:localhost:9998 cori.nersc.gov

Figures: Isaac Henrion
• Again can use default anaconda python:
  
  module load python
  
  python
  
  >>> import torch

• However the anaconda version isn’t built with the pytorch MPI (for multi-node) - so we provide a build
• Again we are looped into intel optimizations

• Below has both those optimizations and MPI

  module load pytorch/v1.0.0-intel
Can use optimized modules by choosing kernels on [jupyter.nersc.gov](http://jupyter.nersc.gov) - e.g.:

- tensorflow-intel(cpu)/1.13.1-py36
- pytorch-v1.1.0

Running on NERSC jupyter hub is normally on a single shared node (so only for smaller models).

Users can deploy distributed deep learning workloads to Cori from Jupyter notebooks using IPyParallel.

- Some examples for running multi-node training and distributed hyper-parameter optimization:
  - [https://github.com/sparticlesteve/cori-intml-examples](https://github.com/sparticlesteve/cori-intml-examples)
Easy Deep learning Example
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import utils as k_utils

# Load MNIST data and add channel
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = np.expand_dims(x_train.astype('float32'), axis=-1)
x_test = np.expand_dims(x_test.astype('float32'), axis=-1)

# normalize data
x_train /= 255
x_test /= 255

num_classes=10

# convert class vectors to binary class matrices
y_train = k_utils.to_categorical(y_train, num_classes)
y_test = k_utils.to_categorical(y_test, num_classes)

Example source: modified version of
github.com/keras-team/keras/blob/master/examples/mnist_cnn.py
Keras: CNN Classification

```python
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
    activation='relu',
    input_shape=[28, 28, 1]))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# compile model
model.compile(loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['accuracy'])

# check model architecture summary
model.summary()
batch_size=128
epochs=5

# train model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
    verbose=1, validation_data=(x_test, y_test))

# evaluate model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0], 'Test accuracy:', score[1])
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_0 (Conv2D)</td>
<td>(None, 26, 26, 32)</td>
<td>320</td>
</tr>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 24, 24, 64)</td>
<td>18496</td>
</tr>
<tr>
<td>max_pooling2d_0 (MaxPooling2D)</td>
<td>(None, 12, 12, 64)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_0 (Dropout)</td>
<td>(None, 12, 12, 64)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_0 ( Flatten)</td>
<td>(None, 9216)</td>
<td>0</td>
</tr>
<tr>
<td>dense_0 (Dense)</td>
<td>(None, 128)</td>
<td>1179776</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 10)</td>
<td>1290</td>
</tr>
</tbody>
</table>

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

Test loss: 0.029
Test accuracy: 0.99
CPU Optimizations and Multi-node training
Tensorflow MKL Optimizations

- Python frameworks rely on optimized backends to perform
- For CPU like Cori KNL this is Intel Math Kernel Library (MKL) (e.g. MKL-DNN) - all recent versions have optimizations
- Blog posts on Intel optimisations:

We track performance via benchmarks. More details at:
- https://docs.nersc.govanalytics/machinlearning/benchmarks/
Multi-node training

- Data parallel training for gradient descent
  - Each node processes data independently then a global update
  - Synchronous; Asynchronous; hybrid gradient lag approaches

- Challenges to HPC scaling have included convergence and performant libraries

From Kurth et al. SC17
arXiv:1708.05256
Performant Multi-Node Libraries

- Initial scaling on NERSC involved a lot of work
  - e.g. with Intel-Caffe and Intel-MLSL
- Default TensorFlow uses gRPC for communication - non-ideal for Cori high-speed network
  - See e.g. Mathuriya et. al (arXiv:1712.09388)
- Fortunately now libraries based on MPI with Horovod and Cray PE ML Plugin

Kurth et al. SC17 arXiv:1708.05256
Kurth et al. Concurrency Computat Pract Exper. 2018;e4989
Horovod Keras Simple Snippets

Available in default NERSC tensorflow modules

• When building model:

```python
from keras import models
import horovod.keras as hvd
model = models.Model(inputs, outputs)
hvd.init()
model.compile(optimizer=hvd.DistributedOptimizer(optimizers.Adam),...)
```

• When training model:

```python
model.fit(callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallback(0),...]
```
Convergence at scale is an active area of research. Some current experiences from multiple projects:

- If strong scaling (small node count): decrease per-node batch size with increasing synchronous node count
- Experiment with increasing learning rate sub-linearly to linearly with number of workers:
  - **Warmup period** starting at initial learning rate
  - Reduce learning rate if learning plateaus
- Advanced Layer-wise adaptive (**LARS/LARC**) strategies
- Be careful with batch normalization for multi-node performance, consider Ghost Batch Normalization
- With ‘global shuffle’ of training examples, **use of Burst Buffer can help with I/O** at NERSC
Support
Contact us

https://docs.nersc.gov/analytics/machinelearning/overview/
General help with deep learning modules; and running DL at NERSC open tickets via: consult@nersc.gov

For collaborations contact ML-Engineers at NERSC:
Mustafa Mustafa: mmustafa@lbl.gov
Steve Farrell: SFarrell@lbl.gov
Wahid Bhimji: wbhimji@lbl.gov