HDF5 and Toy DL for COSINE

Seungmok Lee

2020.02.11

Version Update for IBS Server

- Last week, I visited IBS, created server id, learned COSINE data structure, …
- Version problems occurred again, and I changed my environment to
  - Anaconda 3 → Python 3.7.6
    - keras 2.3.1 | tensorflow 2.1.0 | uproot 3.11.2 | h5py 2.10.0
  - Root 6.14/00
  - No TMVA, no PyRoot

Motivation for Using HDF5

- We will deal with very large training data set, which must exceed our memory capacity.
- So we need a way to load training data set from hard disk, not from memory.

Motivation for Using HDF5 (cont'd)

- There are two ways to load training data from disk:
  - use HDF5 data format supported by h5py module
  - use 'DataGenerator' supported by keras module

HDF5 and DataGenerator

- Using HDF5 is easy, fast and memory conserving.
- While using DataGenerator is difficult, slow and memory consuming.

DataGenerator Performance

```
top - 15:37:37 up  3:34,  1 user,  load average: 9.77, 6.45, 2.91
Tasks: 347 total,  11 running, 252 sleeping,   0 stopped,   0 zombie
%Cpu(s): 94.4 us,  5.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8103580 total,   685436 free,  6806780 used,   611364 buff/cache
KiB Swap:  2097148 total,  2069500 free,    27648 used.   966380 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7700 physmlee  20   0 4096876 630468  18832 R 100.0  7.8   1:57.50 python
 7703 physmlee  20   0 4096876 630456  18776 R  99.7  7.8   2:00.43 python
 7699 physmlee  20   0 4096876 630424  18828 R  99.3  7.8   1:53.32 python
 7702 physmlee  20   0 4096876 630456  18832 R  99.0  7.8   1:52.49 python
 7701 physmlee  20   0 4096876 630456  18832 R  95.3  7.8   1:50.99 python
 7704 physmlee  20   0 4096876 630444  18832 R  95.3  7.8   2:00.46 python
 7698 physmlee  20   0 4096876 630412  18828 R  83.4  7.8   1:57.07 python
 7696 physmlee  20   0 4096876 630480  18832 R  78.4  7.8   1:52.21 python
```

Running MNIST (60K images) with 2 dense layers consumes about 8 * 7.8% * 8GB = 5GB memory and 15 minutes for one epochs.

Note that it took 5 seconds, originally.

```
physmlee@physmlee-desktop:/media/physmlee/sml/research/20022/datagenexer$ python mnist_r
/usr/lib/python2.7/dist-packages/h5py/__init__.py:36: FutureWarning: Conversion of the s
p.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2020-02-11 15:34:42.344044: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CP
2020-02-11 15:34:42.344445: I tensorflow/compiler/xla/service/service.cc:168] XLA servic
2020-02-11 15:34:42.344528: I tensorflow/compiler/xla/service/service.cc:175]   StreamEx
2020-02-11 15:34:42.371400: W tensorflow/compiler/jit/mark_for_compilation_pass.cc:1412]
 want XLA:CPU, either set that envvar, or use experimental_jit_scope to enable XLA:CPU.
) or set the envvar XLA_FLAGS=--xla_hlo_profile.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python2.7/dist-packages/keras/backend/tensorflow_

Epoch 1/5
/usr/local/lib/python2.7/dist-packages/keras/utils/data_utils.py:616: UserWarning: The i
  UserWarning)
 79/468 [====>.....................] - ETA: 13:00 - loss: 0.5823 - accuracy: 0.8262
```

HDF5 Performance

```
top - 15:44:11 up  3:40,  1 user,  load average: 1.71, 2.42, 2.22
Tasks: 316 total,   1 running, 247 sleeping,   0 stopped,   0 zombie
%Cpu(s): 32.8 us,  4.6 sy,  0.0 ni, 59.4 id,  3.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8103580 total,  3956376 free,  2811424 used,  1335780 buff/cache
KiB Swap:  2097148 total,  2069756 free,    27392 used.  4942804 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 8113 physmlee  20   0 3520700 421992 123232 D 255.6  5.2   0:31.16 python
```

Running MNIST with 2 dense layers consumes about 5.2 % * 8GB = 400MB memory and 200 secs for 1.2M data sets. 1.2M data sets are about 1.8 GB!

Note that it took 5 seconds for 1 epoch, originally.

```
physmlee@physmlee-desktop:/media/physmlee/sml/research/20022/MNISTtoHDF$ python MNIST_hdf5.py
/usr/lib/python2.7/dist-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argumen
p.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
2020-02-11 15:46:24.884098: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency:
2020-02-11 15:46:24.884495: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x56367957d
2020-02-11 15:46:24.884528: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device
2020-02-11 15:46:24.920135: W tensorflow/compiler/jit/mark_for_compilation_pass.cc:1412] (One-time wa
 want XLA:CPU, either set that envvar, or use experimental_jit_scope to enable XLA:CPU.  To confirm t
) or set the envvar XLA_FLAGS=--xla_hlo_profile.
WARNING:tensorflow:From /usr/local/lib/python2.7/dist-packages/keras/backend/tensorflow_backend.py:42

Epoch 1/5
1200000/1200000 [==============================] - 202s 168us/step - loss: 0.0432 - accuracy: 0.9865
```

HDF5

- HDF5 format supports saving large, complex, hierarchical data, and fast, easy I/O.
  - Common in science.
  - ‘pip install h5py’ available.
  - Auto-install by uproot
- You can see the tutorials from the link below.

https://www.pythonforthelab.com/blog/how-to-use-hdf5-files-in-python/

HDF5 and Keras

- HDF5 format is easy to feed into Keras.
- Load them using HDF5,

```
# Read dataset from hdf5 file
X_train = HDF5Matrix(filename, 'X_train')
X_test  = HDF5Matrix(filename, 'X_test')
Y_train = HDF5Matrix(filename, 'Y_train')
Y_test  = HDF5Matrix(filename, 'Y_test')
```

- and give them just as they were numpy arrays!

```
# You can give them just as numpy array.
model.fit(X_train, Y_train,
          batch_size=128, epochs=5,
          shuffle = "batch", # You should pass shuffle = "batch" when using HDF5Matrix as input.
          verbose=1)
```

HDF5 in MNIST

- I uploaded two example codes about HDF5 and MNIST on our github.
- One converts MNIST dataset into HDF5 format, and another reads the HDF5 file and trains using that.

COSINE Data Cut

- To test DL on COSINE data, I used the cut below.
  - I used data only from crystal3.

```
// Precut. (3, 4) in https://cupwiki.ibs.re.kr/Kims/EventSelectionforSet2?validation_key=de6e7e670c6bd851c5032ec89a799a37
if( pmtnc31 <=  0  ) continue;
if( pmtnc32 <=  0  ) continue;
if( pmtt131 <= 0.0 ) continue;
if( pmtt132 <= 0.0 ) continue;
if(!(rqcn3  > -1.0)) continue;

// Cut empty waves
if( nclst31 <= 0 ) continue;
if( nclst32 <= 0 ) continue;

// Cut badly clustered waves
if( nclst31 > 15 ) continue;
if (nclst32 > 15 ) continue;

if(lpar > -0.00000000001 && lpar < 0.00000000001) continue; // bad lpar

if(!wantSig) {
        if(lpar > 0.3) continue; // It can be signal. I wnat bkgd.
}
else {
        if(lpar < 0.3) continue; // It can be bkgd. I want signal.
}
```

COSINE Data

- And stored ‘oscil31’ and ‘oscil32’ in a row.

```
Float_t oscil3[4080 * 2];
ntp->SetBranchAddress("oscil31", &oscil3[0]);
ntp->SetBranchAddress("oscil32", &oscil3[4080]);
```

```
treeOut->Branch("oscil3", oscil3, "oscil3[8160]/F");
```

Toy DL for COSINE

- I ran a simple DL for the COSINE data.
  - 33 signals and 1278 backgrounds.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               4178432
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_3 (Activation)    (None, 2)                 0
=================================================================
Total params: 4,442,114
Trainable params: 4,442,114
Non-trainable params: 0
_________________________________________________________________
```

Toy DL Result

- I got 100% !!!

```
Epoch 1/10
1311/1311 [==============================] - 10s 7ms/step - loss: 0.2463 - accuracy: 0.9809
Epoch 2/10
1311/1311 [==============================] - 9s 7ms/step - loss: 0.0510 - accuracy: 0.9886
Epoch 3/10
1311/1311 [==============================] - 9s 7ms/step - loss: 0.0317 - accuracy: 0.9947
Epoch 4/10
1311/1311 [==============================] - 9s 7ms/step - loss: 0.0043 - accuracy: 0.9985
Epoch 5/10
1311/1311 [==============================] - 9s 7ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 6/10
1311/1311 [==============================] - 9s 7ms/step - loss: 2.3830e-04 - accuracy: 1.0000
Epoch 7/10
1311/1311 [==============================] - 9s 7ms/step - loss: 5.4006e-05 - accuracy: 1.0000
Epoch 8/10
1311/1311 [==============================] - 9s 7ms/step - loss: 3.6348e-05 - accuracy: 1.0000
Epoch 9/10
1311/1311 [==============================] - 9s 7ms/step - loss: 2.0508e-05 - accuracy: 1.0000
Epoch 10/10
1311/1311 [==============================] - 9s 7ms/step - loss: 8.5415e-06 - accuracy: 1.0000
```