AI JOURNEY WITH INTEL WORKSHOP

Tuesday, September 17, 12:00 - 2:00 pm
399 Julis Romo Rabinowitz Building
(Free event and lunch is included.)

Why attend:
This workshop will teach you how you can take advantage of your Intel® hardware platforms using with Intel® Optimized frameworks and software offerings without much of a change in your workload.

What you will learn:
This workshop is aimed at providing the attendees a full overview of Intel® AI Software offerings. You will learn how to take advantage of Intel® optimized TensorFlow, and learn the steps of deploying on different hardware with Intel® OpenVino™ Toolkit. Lastly, Intel® DAAL, HPAT, and many other libraries will be introduced and their performance will be shown on a given workload.

Agenda:
✓ Intel® AI Portfolio
✓ AI Journey with Intel®
✓ Intel® Optimized Tensorflow
✓ Intel® AI Libraries
✓ Introduction to Intel® OpenVino™ Toolkit
✓ Optimizations and Performance Comparisons
### CPU TENSORFLOW

**conda create --name tf-cpu --channel anaconda tensorflow**

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Channel</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>3.7.4</td>
<td>h265db76_1</td>
<td>anaconda</td>
</tr>
<tr>
<td>tensorflow</td>
<td>1.14.0</td>
<td>mkl_py37h45c423b</td>
<td>anaconda</td>
</tr>
</tbody>
</table>

**conda create --name tf-cpu --channel intel tensorflow**

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Channel</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>intelpython</td>
<td>2019.5</td>
<td>0</td>
<td>intel</td>
</tr>
<tr>
<td>intel-tensorflow</td>
<td>1.14.0</td>
<td>pypi_0</td>
<td>pypi</td>
</tr>
</tbody>
</table>

**conda create --name IDP --channel intel intelpython3_full**

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Channel</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>daal</td>
<td>2019.5</td>
<td>intel_281</td>
<td>intel</td>
</tr>
<tr>
<td>daal4py</td>
<td>2019.5</td>
<td>py36ha68da19_2</td>
<td>intel</td>
</tr>
<tr>
<td>intelpython</td>
<td>2019.5</td>
<td>0</td>
<td>intel</td>
</tr>
<tr>
<td>intel-tensorflow</td>
<td>1.14.0</td>
<td>pypi_0</td>
<td>pypi</td>
</tr>
</tbody>
</table>
Anaconda vs. Intel Python

(base) [jdh4@della5 ~]$ conda activate tf-cpu-intel
(tf-cpu-intel) [jdh4@della5 ~]$ python
Python 3.6.9 |Intel Corporation| (default, Sep 11 2019, 16:40:08)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Intel(R) Distribution for Python is brought to you by Intel Corporation.
Please check out: https://software.intel.com/en-us/python-distribution

(base) [jdh4@della5 ~]$ module load anaconda3
(base) [jdh4@della5 ~]$ python
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
Intel® Data Analytics Acceleration Library (DAAL)


This library helps reduce the time it takes to develop high-performance data science applications. Enable applications to make better predictions faster and analyze larger data sets with available compute resources.

- **Includes highly optimized machine learning and analytics functions**
- Simultaneously ingests data and computes results for highest throughput performance
- Supports batch, streaming, and distribution use models to meet a range of application needs
- Use the same API for application development on multiple operating systems
- C++, Java and Python APIs
- Multi-threaded
FASTER MACHINE LEARNING
WITH SCIKIT-LEARN AND INTEL® DAAL

SCIKIT-LEARN BENEFITS AND FEATURES

• Scikit-Learn is a mature python package with hundreds of algorithms with different configuration parameters each.

• Intel® DAAL has a different set of algorithms and sometimes implementations use slightly different variants of the algorithm.
import daal4py as d4p
import numpy as np
import pandas as pd
nrows_train = 1000000
nrows_test = 1000000
from time import time

start = time()
print("Reading data set...")
data = pd.read_csv("HIGGS.csv.gz", delimiter="","", header=None, compression=None)
end = time()
print("Reading data set takes \{0\} seconds.".format(end-start))

print("Pre-processing data set...")
start = time()
data = data[list(data.columns[1:]) + list(data.columns[0:1])]
n_features = data.shape[1]
train_data = data[range(1,n_features)][:nrows_train]
train_label = data[range(0, 1)][:nrows_train]
test_data = data[range(1,n_features)][nrows_train:nrows_train+nrows_test]
test_label = data[range(0, 1)][nrows_train:nrows_train+nrows_test]
n_classes = len(np.unique(train_label))
end = time()
print("Pre-processing data set takes \{0\} seconds.".format(end-start))

start = time()
# Configure a training object
train_algo = d4p.gbt_classification_training(**daal_params)
print("Training data set....")
train_result = train_algo.compute(train_data, train_label)
end = time()
print("Training data set takes \{0\} seconds.".format(end-start))

print("Predicting test set...")
start = time()
predict_algo = d4p.gbt_classification_prediction(n_classes)
predict_result = predict_algo.compute(test_data, train_result.model)
end = time()
print("Predicting test data set takes \{0\} seconds.".format(end-start))
Gradient Boosting training - speedup vs. XGBoost CPU baseline 0.81 (Higher is better)
conda create --name intel-sklearn scikit-learn -c intel

<table>
<thead>
<tr>
<th>package</th>
<th>build</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2-1.0.6</td>
<td>18</td>
</tr>
<tr>
<td>certifi-2018.1.18</td>
<td>py36_2</td>
</tr>
<tr>
<td>daal-2019.5</td>
<td>intel_281</td>
</tr>
<tr>
<td>daal4py-2019.5</td>
<td>py36ha68da19_2</td>
</tr>
<tr>
<td>icc_rt-2019.5</td>
<td>intel_281</td>
</tr>
<tr>
<td>interp-rt-2019.5</td>
<td>intel_281</td>
</tr>
<tr>
<td>intel-openmp-2019.5</td>
<td>intel_281</td>
</tr>
<tr>
<td>intelpython-2019.5</td>
<td>0</td>
</tr>
<tr>
<td>joblib-0.13.2</td>
<td>py36_1</td>
</tr>
<tr>
<td>mkl-2019.5</td>
<td>intel_281</td>
</tr>
<tr>
<td>mkl-service-2.3.0</td>
<td>py36_0</td>
</tr>
<tr>
<td>mkl_fft-1.0.14</td>
<td>py36ha68da19_1</td>
</tr>
<tr>
<td>mkl_random-1.0.4</td>
<td>py36ha68da19_2</td>
</tr>
<tr>
<td>numpy-1.17.0</td>
<td>py36ha68da19_13</td>
</tr>
<tr>
<td>numpy-base-1.17.0</td>
<td>py36_13</td>
</tr>
<tr>
<td>openssl-1.0.2s</td>
<td>0</td>
</tr>
<tr>
<td>pip-19.1.1</td>
<td>5</td>
</tr>
<tr>
<td>python-3.6.9</td>
<td>5</td>
</tr>
<tr>
<td>scipy-1.3.1</td>
<td>py36ha68da19_4</td>
</tr>
<tr>
<td>setuptools-41.0.1</td>
<td>py36_0</td>
</tr>
<tr>
<td>six-1.12.0</td>
<td>py36_0</td>
</tr>
<tr>
<td>sqlite-3.28.0</td>
<td>py36_0</td>
</tr>
<tr>
<td>tbb-2019.8</td>
<td>intel_281</td>
</tr>
<tr>
<td>tbb4py-2019.8</td>
<td>py36_intel_0</td>
</tr>
<tr>
<td>tcl-8.6.4</td>
<td>24</td>
</tr>
<tr>
<td>tk-8.6.4</td>
<td>29</td>
</tr>
<tr>
<td>wheel-0.31.0</td>
<td>5</td>
</tr>
<tr>
<td>xz-5.2.4</td>
<td>5</td>
</tr>
<tr>
<td>zlib-1.2.11</td>
<td>5</td>
</tr>
</tbody>
</table>
// import org.apache.spark.mllib.feature.{PCA, PCAModel}
import daal_for_mllib.{PCA, PCAModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Matrix
import org.apache.spark.mllib.linalg.distributed.RowMatrix
val data = sc.textFile("/Spark/PCA/data/PCA.txt")
val dataRDD = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()
val model = new PCA(10).fit(dataRDD)
println("Principal components:" + model.pc.toString())
Intel® High Performance Analytics Toolkit (HPAT)

**A compiler-based framework for big data in Python**

High Performance Analytics Toolkit (HPAT) scales analytics/ML codes in Python to bare-metal cluster/cloud performance automatically. It compiles a subset of Python (Pandas/Numpy) to efficient parallel binaries with MPI, requiring only minimal code changes. HPAT is orders of magnitude faster than alternatives like Apache Spark.

[https://github.com/IntelPython/hpat](https://github.com/IntelPython/hpat)
HPAT knows how to handle pandas!

```python
In [1]:
import hpat
import pandas as pd
import numpy as np

@hpat.jit
def calc_pi(n):
    xy = pd.DataFrame({'x': 2 * np.random.rand(n) - 1,
                       'y': 2 * np.random.rand(n) - 1})
    pi = 4 * xy[xy.x**2 + xy.y**2 < 1].x.count() / n
    return pi

%timeit calc_pi(2**22)
```

77.6 ms ± 5.29 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
import hpat
import numpy as np
import time

@hpat.jit
def calc_pi(n):
    t1 = time.time()
    x = 2 * np.random.ranf(n) - 1
    y = 2 * np.random.ranf(n) - 1
    pi = 4 * np.sum(x**2 + y**2 < 1) / n
    print("Execution time:", time.time()-t1, "\nresult:", pi)
    return pi

calc_pi(2 * 10**8)

mpiexec -n 8 python pi.py
SUMMARY

• Consider installing TensorFlow from the Intel channel if running on CPU-only machines
• MKL-DNN is now DNNL
• Scikit-Learn and Spark users may benefit from DAAL
• Pandas users may benefit from HPAT
• Intel XGBoost