Deep Learning Pipelines for High Energy Physics using Apache Spark with Distributed Keras and Analytics Zoo

Luca Canali, CERN

#UnifiedDataAnalytics #SparkAISummit
About Luca

• Data Engineer at CERN
  – Hadoop and Spark service, database services
  – 19+ years of experience with data engineering

• Sharing and community
  – Blog, notes, tools, contributions to Apache Spark

@LucaCanaliDB – http://cern.ch/canali
CERN:
Particle Accelerators (LHC)
High Energy Physics Experiments
Experimental High Energy Physics is Data Intensive

Particle Collisions

Physics Discoveries

Large Scale Computing

https://twiki.cern.ch/twiki/pub/CMSPublic/Hig13002TWiki/HZZ4l_animated.gif

And https://iopscience.iop.org/article/10.1088/1742-6596/455/1/012027
Key Data Processing Challenge

- Proton-proton collisions at LHC experiments happen at 40MHz.
  - Hundreds of TB/s of electrical signals that allow physicists to investigate particle collision events.
- Storage, limited by bandwidth
  - Currently, only 1 every ~40K events stored to disk (~10 GB/s).
Data Flow at LHC Experiments

This can generate up to a **petabyte** of raw data per second
Reduced to **GB/s** by filtering in real time
Key is how to select potentially interesting events (**trigger** systems).
R&D – Data Pipelines

- Improve the **quality of filtering systems**
  - Reduce false positive rate
  - From rule-based algorithms to classifiers based on Deep Learning

- Advanced analytics **at the edge**
  - Avoid wasting resources in offline computing
  - Reduction of operational costs
Particle Classifiers Using Neural Networks

- **R&D to improve the quality of filtering systems**
  - Develop a “Deep Learning classifier” to be used by the filtering system
  - **Goal:** Identify events of interest for physics and reduce false positives
    - False positives have a **cost**, as wasted storage bandwidth and computing
Deep Learning Pipeline for Physics Data

1. Data Ingestion
   - Read physics data and feature engineering

2. Feature Preparation
   - Prepare input for Deep Learning network

3. Model Development
   - 1. Specify model topology
   - 2. Tune model topology on small dataset

4. Training
   - Train the best model

Analytics Platform at CERN

Integrating new “Big Data” components with existing infrastructure:
- **Software** distribution
- **Data** platforms

- **HEP software**
- **Experiments storage**
- **HDFS**
- **Personal storage**
Do the heavy lifting in spark and collect aggregated view to panda DF

```python
In [11]:
df_loadavg_pandas = spark.sql(~SELECT submitter_host, 
                            avg(body.LoadAvg) as avg, 
                            hour(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')) as hr 
                            FROM loadavg 
                            WHERE submitter_hostgroup = 'hadoop/ifdb/datanode' 
                            AND dayofmonth(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')) = 15 
                            GROUP BY hour(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')), submitter_host'
                            .toPandas()
```

Visualize with seaborn

```python
In [10]:

# heatmap of service availability
gf = plt.figure(figsize=(10, 6))
ax = sns.heatmap(df_loadavg_pandas.pivot(index='submitter_host', columns='hr', values='avg'), cmap='Blues')
ax.set_title('Heatmap of loadAvg')
```

Out[10]:
Text(0.5, 1.0, 'Heatmap of loadAvg')
Hadoop and Spark Clusters at CERN

- Clusters:
  - YARN/Hadoop
  - Spark on Kubernetes
- Hardware: Intel based servers, continuous refresh and capacity expansion

<table>
<thead>
<tr>
<th>Accelerator logging (part of LHC infrastructure)</th>
<th>Hadoop - YARN - 30 nodes (Cores - 1200, Mem - 13 TB, Storage – 7.5 PB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose</td>
<td>Hadoop - YARN, 65 nodes (Cores – 2.2k, Mem – 20 TB, Storage – 12.5 PB)</td>
</tr>
<tr>
<td>Cloud containers</td>
<td>Kubernetes on Openstack VMs, Cores - 250, Mem – 2 TB Storage: remote HDFS or EOS (for physics data)</td>
</tr>
</tbody>
</table>
Extending Spark to Read Physics Data

- **Physics data**
  - Currently: >300 PBs of Physics data, increasing ~90 PB/year
  - Stored in the CERN EOS storage system in ROOT Format and accessible via XRootD protocol

- **Integration with Spark ecosystem**
  - Hadoop-XRootD connector, HDFS compatible filesystem
  - Spark Datasource for ROOT format

https://github.com/cerndb/hadoop-xrootd
https://github.com/diana-hep/spark-root
Labeled Data for Training and Test

- Simulated events
  - Software simulators are used to generate events and calculate the detector response
  - Raw data contains arrays of simulated particles and their properties, stored in ROOT format
  - 54 million events
Step 1: Data Ingestion

- **Read input files:** 4.5 TB from custom (ROOT) format
- **Feature engineering**
  - Python and PySpark code, using Jupyter notebooks
- **Write output in Parquet format**

**Input:**
- 54 M events ~4.5 TB
- Physics data storage (EOS)
- Physics data format (ROOT)

**Output:**
- 25 M events
- 950 GB in Parquet format
- Target storage (HDFS)
Feature Engineering

- Filtering
  - Multiple filters, keep only events of interest
  - Example: “events with one electrons or muon with Pt > 23 Gev”

- Prepare “Low Level Features”
  - Every event is associated to a matrix of particles and features (801x19)

```python
features = [
    'Energy', 'Px', 'Py', 'Pz', 'Pt', 'Eta', 'Phi',
    'vtxX', 'vtxY', 'vtxZ', 'ChPFIso', 'GammaPFIso', 'NeuPFIso',
    'isChHad', 'isNeuHad', 'isGamma', 'isEle', 'isMu', 'Charge'
]
```

- High Level Features (HLF)
  - Additional 14 features are computed from low level particle features
  - Calculated based on domain-specific knowledge
Step 2: Feature Preparation

Features are converted to formats suitable for training

- One Hot Encoding of categories
- MinMax scaler for High Level Features
- Sorting Low Level Features: prepare input for the sequence classifier, using a metric based on physics. This use a Python UDF.
- Undersampling: use the same number of events for each of the three categories

Result

- 3.6 Million events, 317 GB
- Shuffled and split into training and test datasets
- Code: in a Jupyter notebook using PySpark with Spark SQL and ML
Performance and Lessons Learned

- Data preparation is **CPU bound**
  - Heavy serialization-deserialization due to Python UDF
- Ran using 400 cores: data ingestion took ~3 hours,
- It can be **optimized**, but is it worth it?
  - Use Spark SQL, Scala instead of Python UDF
  - Optimization: replacing parts of Python UDF code with Spark SQL and **higher order functions**: run time from 3 hours to 2 hours

```sql
FILTER(Electron,
    electron -> electron.PT > 23
) Electron,
FILTER(MuonTight,
    muon -> muon.PT > 23
) MuonTight
```

```sql
WHERE cardinality(Electron) > 0
OR cardinality(MuonTight) > 0
```
Neural Network Models and

1. Fully connected feed-forward deep neural network
   • Trained using High Level Features (~1 GB of data)

2. Neural network based on Gated Recurrent Unit (GRU)
   • Trained using Low Level Features (~300 GB of data)

3. Inclusive classifier model
   • Combination of (1) + (2)
Hyper-Parameter Tuning – DNN

• Hyper-parameter tuning of the DNN model
  • Trained with a subset of the data (cached in memory)
  • Parallelized with Spark, using spark_sklearn.grid_search
    • And scikit-learn + keras: tensorflow.keras.wrappers.scikit_learn
Deep Learning at Scale with Spark

- Investigations and constraints for our exercise
- How to run deep learning in a Spark data pipeline?
  - Neural network models written using Keras API
  - Deploy on Hadoop and/or Kubernetes clusters (CPU clusters)
- Distributed deep learning
  - GRU-based model is complex
  - Slow to train on a single commodity (CPU) server
Spark, Analytics Zoo and BigDL

- **Apache Spark**
  - Leading tool and API for data processing at scale

- **Analytics Zoo** is a platform for **unified** analytics and AI
  - Runs on Apache Spark leveraging BigDL / Tensorflow
  - For service developers: integration with infrastructure (hardware, data access, operations)
  - For users: Keras APIs to run user models, integration with Spark data structures and pipelines

- **BigDL** is an open source distributed deep learning framework for Apache Spark
BigDL Run as Standard Spark Programs

**Standard Spark jobs**
- No changes to the Spark or Hadoop clusters needed

**Iterative**
- Each iteration of the training runs as a Spark job

**Data parallel**
- Each Spark task runs the same model on a subset of the data (batch)

Source: Intel BigDL Team
BigDL Parameter Synchronization

Source: https://github.com/intel-analytics/BigDL/blob/master/docs/docs/whitepaper.md
Model Development – DNN for HLF

• Model is instantiated using the Keras-compatible API provided by Analytics Zoo

```python
In [7]:
# Create kers like zoo model.
# Only need to change package name from keras to zoo.pipeline.api.keras

from zoo.pipeline.api.keras.optimizers import Adam
from zoo.pipeline.api.keras.models import Sequential
from zoo.pipeline.api.keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(50, input_shape=(14,), activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))

creating: createZooKerasSequential
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
```
Model Development – GRU + HLF

A more complex network topology, combining a GRU of Low Level Feature + a DNN of High Level Features

```python
from zoo.pipeline.api.keras.optimizers import Adam
from zoo.pipeline.api.keras.models import Sequential
from zoo.pipeline.api.keras.layers.core import *
from zoo.pipeline.api.keras.layers.recurrent import GRU
from zoo.pipeline.api.keras.engine.topology import import Merge

## GRU branch
gruBranch = Sequential()
    .add(Masking(0.0, input_shape=(801, 19)))
    .add(GRU(output_dim=50,
             activation='tanh'))
    .add(Dropout(0.2))

## HLF branch
hlfBranch = Sequential()
    .add(Dropout(0.2, input_shape=(14,)))

## Concatenate the branches
branches = Merge(layers=[gruBranch, hlfBranch], mode='concat')

## Create the model
model = Sequential()
    .add(branches)
    .add(Dropout(0.2))
    .add(Dense(25, activation='relu'))
    .add(Dense(3, activation='softmax'))
```

## Diagram

![Diagram of GRU + HLF network topology](image_url)
Distributed Training

Instantiate the estimator using Analytics Zoo / BigDL

```python
# Create SparkML compatible estimator for deep learning training
from bigdl.optim.optimizer import EveryEpoch, Loss, TrainSummary, ValidationSummary
from zoo.pipeline.nnframes import *
from zoo.pipeline.api.keras.objectives import CategoricalCrossEntropy

estimator = NNEstimator(model, CategoricalCrossEntropy())
    .setOptimMethod(Adam())
    .setBatchSize(BDLbatch)
    .setMaxEpoch(numEpochs)
    .setFeaturesCol("MLF_input")
    .setLabelCol("encoded_label")
    .setValidation(trigger=EveryEpoch(), val_df=testDF,
                  val_method=[loss=CategoricalCrossEntropy()]
                   , batch_size=BDLbatch)
```

The actual training is distributed to Spark executors

```
%%time
trained_model = estimator.fit(trainDF)
```

Storing the model for later use

```python
modelDir = logDir + '/nnmodels/MLFClassifier'
trained_model.save(modelDir)
```
Performance and Scalability of Analytics Zoo/BigDL

Analytics Zoo/BigDL on Spark scales up in the ranges tested

Inclusive classifier model

DNN model, HLF features
Workload Characterization

- Training with Analytics zoo
  - GRU-based model: Distributed training on YARN cluster
  - Measure with Spark Dashboard: it is CPU bound
Results – Model Performance

- Trained models with Analytics Zoo and BigDL
- Met the expected results for model performance: ROC curve and AUC
Training with TensorFlow 2.0

- Training and test data
  - Converted from Parquet to TFRecord format using Spark
  - TensorFlow: data ingestion using tf.data and tf.io
- Distributed training with tf.distribute + tool for K8S: https://github.com/cerndb/tf-spawner

Distributed training with TensorFlow 2.0 on Kubernetes (CERN cloud)

Distributed training of the Keras model with:
tf.distribute.experimental.MultiWorkerMirroredStrategy
Performance and Lessons Learned

• Measured distributed training elapsed time
  • From a few hours to 11 hours, depending on model, number of epochs and batch size. Hard to compare different methods and solutions (many parameters)

• Distributed training with BigDL and Analytics Zoo
  • Integrates very well with Spark
  • Need to cache data in memory
  • Noisy clusters with stragglers can add latency to parameter synchronization

• TensorFlow 2.0
  • It is straightforward to distribute training on CPUs and GPUs with tf.distribute
  • Data flow: Use TFRecord format, read with TensorFlow’s tf.data and tf.io
  • GRU training performance on GPU: 10x speedup in TF 2.0
  • Training of the Inclusive Classifier on a single P100 in 5 hours
Recap: our Deep Learning Pipeline with Spark

Input: labeled data and DL models

Feature engineering at scale

Hyperparameter optimization (Random/Grid search)

Distributed model training

Output: particle selector model
Model Serving and Future Work

- Using Apache Kafka and Spark?

- FPGA serving DNN models

Output pipeline: to storage / further online analysis
Summary

• The use case developed addresses the needs for higher efficiency in event filtering at LHC experiments
• Spark, Python notebooks
  • Provide well-known APIs and productive environment for data preparation
• Data preparation performance, lessons learned:
  • Use Spark SQL/DataFrame API, avoid Python UDF when possible
• Successfully scaled Deep Learning on Spark clusters
  • Using Analytics Zoo and BigDL
  • Deployed on existing Intel Xeon-based servers: Hadoop clusters and cloud
• Good results also with Tensorflow 2.0, running on Kubernetes
• Continuous evolution and improvements of DL at scale
  • Data preparation and scalable distributed training are key
Acknowledgments

• Matteo Migliorini, Marco Zanetti, Riccardo Castellotti, Michał Bień, Viktor Khristenko, CERN Spark and Hadoop service, CERN openlab
• Authors of “Topology classification with deep learning to improve real-time event selection at the LHC”, notably Thong Nguyen, Maurizio Pierini
• Intel team for BigDL and Analytics Zoo: Jiao (Jennie) Wang, Sajan Govindan
  – Analytics Zoo: https://github.com/intel-analytics/analytics-zoo
  – BigDL: https://software.intel.com/bigdl

References:

– Data and code: https://github.com/cerndb/SparkDLTrigger
DON’T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

Download on the App Store
GET IT ON Google Play