Bigdl: A Distributed Deep Learning Framework For Big Data

Pouya Kousha
Fall 2018 – CSE 5194
Prof. DK Panda
Agenda Of Presentation

- Motivation And Intro
- Programming Model
  - Spark
  - Data Transformation
  - Model Construction
  - Model Training
  - Model Inference
- Execution Model
  - Data Parallel Training
  - Parameter Synchronization
  - Task Scheduling
  - Model Quantization
  - Local Execution
- Application
  - Image Feature Extraction
  - Precipitation Nowcasting
  - Image-similarity Based House Recommendations
What’s BigDL?

- **BigDL** is a distributed deep learning library for Apache Spark
- Users can write their DL applications as standard Spark programs running on the top of Spark/Hadoop clusters
- Comes to existence due to emerging convergence of Big data and DL
- Rich DL support
  - Can load pre-trained Caffe, Torch, Keras models
  - Neural network operation
  - Layers, losses and optimizations
  - Users can run existing models on top of Spark
You may want to write your deep learning programs using BigDL if:

- You want to analyze a large amount of data on the same Big Data (Hadoop/Spark) cluster where the data are stored (in, say, HDFS, HBase, Hive, etc.).

- You want to add deep learning functionalities (either training or prediction) to your Big Data (Spark) programs and/or workflow.

- You want to leverage existing Hadoop/Spark clusters to run your deep learning applications, which can be then dynamically shared with other workloads (e.g., ETL, data warehouse, feature engineering, classical machine learning, graph analytics, etc.)

Source: https://bigdl-project.github.io/0.7.0/
Programming Model: Spark

- Built on the top of Apache Spark
- Provides unified programming paradigm
- User can load very large datasets and process the data using Spark
- Use the data processed by Spark to the integrated data-analytics DL and AI pipelines

```python
1    spark = SparkContext(appName="text_classifier", ...)
2    #load input data: (text, label) pairs
3    texts_rdd = spark.textFile("hdfs://...")
4    #convert text to list of words
5    words_rdd = texts_rdd.map(lambda text, label:
6                                      ([w for w in to_words(text)], label))
```
Programming Model: Data Transformation

- Spark supports DAG by composing multiple data-parallel operators on RDD
  - Users can transform the input data by constructing the dataflow DAG in Spark
- The transformed data can be used by the Neural Network (NN) models.
- **N-dimensional array**: Models the basic data elements on NN computations as N-dimensional arrays
- **Sample**: Each record used in BigDL model training and prediction is modelled as sample
  - Input feature (one or more N-dimensional array)
  - Optional label (one or more N-dimensional array | scalar value)

```java
7    // load GloVe embedding
8    w2v = news20.get_glove_w2v(dim=...)
9    // convert word list to list of vectors using GloVe embeddings
10   vector_rdd = words_rdd.map(lambda word_list, label:
11       ([to_vec(w, w2v) for w in word_list], label))
12   // convert (list of vectors, label) pair to Sample
13   sample_rdd = vector_rdd.map(lambda vector_list, label:
14                                to_sample(vector_list, label))
```
Programming Model: Model Construction

- BigDL uses a dataflow representation for NN model where each vertex in the dataflow graph represents a NN layer
  - Similar to Caffe, Torch and Keras

- BigDL uses the semantics of the layers for model evaluation (forward) and gradient computation (backward)

```python
15   // construct neural network model
16   model = Sequential().add(Recurrent().add(LSTM(...)))
17       .add(Linear(...))
18       .add(LogSoftMax())
```
Programming Model: Model Training

- RDD of samples (transformed data) and constructed model will be passed to Optimizer
- **Task of Optimizer**: automatically perform distributed model training across cluster
  - Runs multiple iterative Spark jobs to minimize the loss using an optimization method
- **Visualizer**: to make it easy for users to understand, the optimizer can produce a TrainSummary that contains various data (loss, weight, etc.)
  - Can be visualized in TensorBoard or Jupyter Notebooks

```java
19 // train the model
20 loss = ClassNLLCriterion()
21 optim_method = Adagrad()
22 optimizer = Optimizer(model=model, training_rdd=sample_rdd,
23                        criterion=loss, optim_method=optim_method, ...)
24 optimizer.set_train_summary(summary = TrainSummary(...))
25 trained_model = optimizer.optimize()
```
Programming Model: Model Inference

- BigDL let users to utilize existing pre-trained DL models in Spark
  - TensorFlow
  - Caffe
  - Keras
  - Torch
  - BigDL

- **Model Broadcast**: BigDL provides a ModelBroadcast abstraction to manage the deployment of the pre-trained model across the cluster in a Spark job

```java
26 //model prediction
27 test_rdd = ...
28 prediction_rdd = trained_model.predict(test_rdd)
```
Execution Model

- In Spark, RDDs are transformed through **coarse-grained** operators
  - Apply the same operation to all data items >> bottleneck
- Efficient and distributed DNN requires **fine-grained data access and in-place data mutation**

How does BigDL address efficient and scalable distributed training on top of Spark?
Data parallel training

- Using synchronous mini-batch SGD, BigDL provides data parallel training on spark
  - Efficient
  - Scalable

- Implemented as iterative process
  - Each iteration runs a couple of Spark jobs (compute gradients using mini-batches)
  - Then make a single update to the parameters of neural network model

- BigDL co-partition and co-locate two types of RDDs
  - The training data as RDD of samples
  - Models
In each iteration of the model training, a model forward-backward computes the local gradients for each model replica.
Parameter Synchronization

- A performance critical operation for data-parallel training
- Existing DL frameworks usually implement parameter server architecture on AllReduce
  - Cannot be supported by functional compute model in Big Data systems
- Solution: adapt primitives in Spark to implement a similar efficient AllReduce semantic
  - Same functionality of parameter server in other frameworks
Both shuffle and broadcast are implemented on top of distributed in-memory storage in Spark.

For each task $n$ in the "parameter synchronization" job:
- Shuffle the $n^{th}$ partition of all gradients to this task.
- Aggregate (sum) the gradients.
- Update the $n^{th}$ partition of the weights.
- Broadcast the $n^{th}$ partition of the updated weights.

Tasks in the model forward-backward job of the next iteration can read the latest value of all the weights before the next training step begins.

AllReduce operation in BigDL
Scaling BigDL AllReduce operation

- By having AllReduce operation, BigDL provides a efficient scalable parameter server on top of Big Data frameworks.
- Throughput of ImageNet Inception v1 training reported by Cray using BigDL 0.3.0

The training throughput scales up almost linearly.
Task Scheduling in BigDL

- Similar functionality to parameter server but different implementation

- Benefits
  - Running a series of short-lived spark jobs (two jobs per mini-batch)
  - Each task in the job is stateless and non-blocking
  - Adapts to the dynamic resource changes

- Problem
  - Task scheduling in Spark can become a bottleneck on large cluster

- Quick fix
  - Launch a single multi-threaded task on each worker to achieve high scalability
  - Use group scheduling for multiple iterations of computation (Drizzle)
Effect of group scheduling on scaling tasks

- Overheads of task scheduling and dispatch (as a fraction of average compute time) for ImageNet Inception v1 in BigDL.
Model Quantization

- Refers to performing calculation/storing for numbers in more compact and lower precision form.

- BigDL performs local quantization
  - In each local quantization window (sub block of patch or kernel)

- Local quantization
  - Load the model and quantize parameters into 8-bit integer
  - During model inference, quantize input data into 8-bit integer on the fly
  - Apply computation in the quantized model
  - Dequantize the results into float32
Local Execution

- Enables running training and inference on a local JVM machine (no Spark)
- Improves efficiency when running on a single node
- Easier to integrate BigDL models with various big data frameworks
  - Apache Storm
  - Apache Flink
  - Apache Kafka
Applications

- In the next slides, some applications are studied
  - Model Inference: image feature extraction
  - Distributed Training: precipitation nowcasting
  - Transfer Learning: image-similarity based house recommendations
Generate target images by resize and crop

Use BigDL to load SDD model for object detection in Spark

Read images from database into Spark and pre-process the RDDs of pics

Use BigDL to load DeepBit model for distributed feature extraction of the target images to generate features and save them

Model Inference for image feature extraction

Figure 11. End-to-end object detection and image feature extraction pipeline (using SSD and DeepBit models) on top of Spark and BigDL [42].

Throughput of GPU clusters and Xeon clusters for the image feature extraction pipeline benchmarked by JD; the GPU throughput is tested on 20 NVIDIA Tesla K40 cards, and the Xeon throughput is tested on 1200 logical cores.
Distributed Training: precipitation nowcasting

Figure 13. End-to-end precipitation nowcasting workflow (using sequence-to-sequence model) on Spark and BigDL [37].

Figure 14. Predicting precipitation patterns for the next hour (i.e., a sequence of images for the future time steps of the next hour) on Spark and BigDL [37].
Conclusion

- Described BigDL (programming model, execution model and use cases)
- BigDL combines the benefits of Big Data and HPC architecture
  - Benefits the data scientist who are not familiar with DL
  - Data-analytics integrated deep learning programming model
  - Highly efficient and scalable parameter-server on top of Big Data platform
- Currently v0.7.0 is available at https://bigdl-project.github.io
- BigDL can be seen as one of many possible proof-of-concept applications that support Intel’s plans in war with GPUs taking over the market