Building Deep Learning Applications for Big Data

An Introduction to Analytics Zoo: Distributed TensorFlow, Keras and BigDL on Apache Spark

Jason Dai
Agenda

• Motivation (15 minutes)
  • Trends, real-world scenarios

• DL frameworks on Apache Spark (30 minutes)
  • BigDL, TensorFlowOnSpark, DL Pipelines, Project Hydrogen, SparkNet

• Analytics Zoo (30 minutes)
  • Distributed TensorFlow, Keras and BigDL on Apache Spark

• Analytics Zoo Examples (30 minutes)
  • Dogs vs. cats, object detection, OpenVINO model inference, distributed TensorFlow

• Break (30 minutes)
Agenda

• Distributed training in BigDL (30 minutes)
  • Data parallel training, parameter synchronization, scaling & convergence, etc.

• Advanced applications (20 minutes)
  • Text classification, movie recommendation

• Real-world applications (45 minutes)
  • Object detection and image feature extraction at JD.com
  • Produce defect detection using distributed TF on Spark in Midea
  • NLP based customer service chatbot for Microsoft Azure
  • Image similarity based house recommendation for MLSlisting
  • Transfer learning based image classifications for World Bank
  • LSTM-Based time series anomaly detection for Baosight
  • Fraud detection for payment transactions for UnionPay

• Conclusion and Q&A (10 minutes)
Motivations

Technology and Industry Trends

Real World Scenarios
Trend #1: Data Scale Driving Deep Learning Process

“Machine Learning Yearning”, Andrew Ng, 2016
Trend #2: Hadoop Becoming the Center of Data Gravity

Why an Enterprise Data Hub?

- Single place for all enterprise data... (unedited hi-resolution history of everything)
- Reduces Application Integration Costs
  - Connect once to Hub (N vs N² connections)
- Lowest unit cost data processing & storage platform
  - Open source S/W on commodity H/W (reliability in S/W not H/W)
  - Can mix H/W vendors means every expansion is competitively tendered
- Fast Standardised Provision
  - No custom design task, re-use Active Directory account/password processes
  - Reduces Shadow IT
- Secure (audited, E2E visibility/auditing, encryption)
  - Eliminate need for one off extracts

Everyone is building Data Lakes

- Universal data acquisition makes all big data analytics and reporting easier
- Hadoop provides a scalable storage with HDFS
- How will we scale consumption and curation of all this data?

Phillip Radley, BT Group
Strata + Hadoop World 2016 San Jose

Matthew Glickman, Goldman Sachs
Spark Summit East 2015
Trend #3: Real-World ML/DL Systems Are Complex Big Data Analytics Pipelines

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Trend #4: Unified Big Data Platform Driving Analytics & Data Science

Ion Stoica, UC Berkeley, Spark Summit 2013 Keynote
Unified Big Data Analytics Platform

Apache Hadoop & Spark Platform

- Data Processing & Analysis
  - Batch
  - Streaming
  - Interactive
  - DataFrame
    - SQL
    - SparkR
    - Streaming
    - ML Pipelines
      - MLlib
      - GraphX
    - Spark Core
  - Machine Learning
  - Graph Analytics
  - SQL
  - Notebook
  - Resource Mgmt & Co-ordination
    - YARN
    - ZooKeeper
  - Data Input
    - Flume
    - Kafka
  - Storage
    - HDFS
    - Parquet
    - Avro
    - HBase
  - Spark Core
    - Flink
    - Storm
    - MR
    - Giraph
  - Java
  - Python
  - R
  - Notebook
  - Spreadsheet

Notebook
Spreadsheet
Chasm b/w Deep Learning and Big Data Communities

Deep learning experts

The Chasm

Average users (big data users, data scientists, analysts, etc.)
Large-Scale Image Recognition at JD.com
Bridging the Chasm

Make deep learning more accessible to big data and data science communities

- Continue the use of familiar SW tools and HW infrastructure to build deep learning applications

- Analyze “big data” using deep learning on the same Hadoop/Spark cluster where the data are stored

- Add deep learning functionalities to large-scale big data programs and/or workflow

- Leverage existing Hadoop/Spark clusters to run deep learning applications
  - Shared, monitored and managed with other workloads (e.g., ETL, data warehouse, feature engineering, traditional ML, graph analytics, etc.) in a dynamic and elastic fashion
DL Frameworks on Apache Spark

BigDL, DL Pipelines for Spark, TensorflowOnSpark, Project Hydrogen of Spark, SparkNet, etc.
A Spark cluster consists of a single *driver* node and multiple *worker* nodes.

A Spark *job* contains many Spark *tasks*, each working on a data *partition*.

Driver is responsible for scheduling and dispatching the tasks to workers, which runs the actual Spark tasks.

[https://spark.apache.org](https://spark.apache.org)
Apache Spark
Spark Program

- Spark runs as a library in your program (1 instance per app)
- Runs tasks locally or on cluster
  - K8s, YARN, Mesos or standalone mode
- Accesses storage systems via Hadoop InputFormat API
  - Can use HBase, HDFS, S3, ...

Source: “Parallel programming with Spark”, Matei Zaharia, AMPCamp 3
Apache Spark
Distributed Task Execution

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles

Source: “Parallel programming with Spark”, Matei Zaharia, AMPCamp 3
BigDL
Bringing Deep Learning To Big Data Platform

- **Distributed** deep learning framework for Apache Spark*
- Make deep learning more accessible to **big data users** and **data scientists**
  - Write deep learning applications as **standard Spark programs**
  - Run on existing Spark/Hadoop clusters (**no changes needed**)
- Feature parity with popular deep learning frameworks
  - E.g., Caffe, Torch, Tensorflow, etc.
- High performance (on CPU)
  - Powered by Intel MKL and multi-threaded programming
- Efficient scale-out
  - Leveraging Spark for distributed training & inference

[https://github.com/intel-analytics/BigDL](https://github.com/intel-analytics/BigDL)
[https://bigdl-project.github.io/](https://bigdl-project.github.io/)
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)
Distributed Training in BigDL

Parameter Server Architecture
directly inside Spark (using Block Manager)

Peer-2-Peer All-Reduce synchronization
Load existing TF or Keras models in Spark ML pipelines

- Load into transformer: inference only
- Load into estimator: single node training/tuning only


https://github.com/databricks/spark-deep-learning
**TensoflowOnSpark**

**Standalone TF jobs on Spark cluster**
- Use Spark as the orchestration layer to allocate resources
- Launch distributed TensorFlow job on the allocated resources
- Coarse-grained integration of two independent frameworks
  - Memory overheads, no gang scheduling, limited interactions with data pipelines, etc.

`feed_dict`: TF worker func runs as independent process in background, reading data from Python queue

`queue_runner`: direct HDFS access from TF work func

https://github.com/yahoo/TensorFlowOnSpark
Project Hydrogen

Different execution models

Spark
- Tasks are independent of each other
- Embarrassingly parallel & massively scalable
- If one crashes, rerun that one

Distributed Training
- Complete coordination among tasks
- Optimized for communication
- If one crashes, must rerun all tasks

Spark and distributed TF have different execution model
- Support “gang scheduling” through new barrier execution mode

Overhead of data transferring between Spark and TF
- Optimized data exchange leveraging Apache Arrow

Distributed DL training by running Caffe in each worker

- Asynchronous parameter synchronization through master (driver) mode
  - Very inefficient (~20 seconds with just 5 workers)

Analytics Zoo

A unified analytics + AI platform for distributed TensorFlow, Keras and BigDL on Apache Spark

https://github.com/intel-analytics/analytics-zoo
# Analytics Zoo

Unified Analytics + AI Platform for Big Data

## Distributed TensorFlow, Keras and BigDL on Spark

<table>
<thead>
<tr>
<th>Reference Use Cases</th>
<th>• Anomaly detection, sentiment analysis, fraud detection, image generation, chatbot, etc.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Built-In Deep Learning Models</th>
<th>• Image classification, object detection, text classification, text matching, recommendations, sequence-to-sequence, anomaly detection, etc.</th>
</tr>
</thead>
</table>

| Feature Engineering | Feature transformations for  
<table>
<thead>
<tr>
<th></th>
<th>• Image, text, 3D imaging, time series, speech, etc.</th>
</tr>
</thead>
</table>

| High-Level Pipeline APIs | • Distributed TensorFlow and Keras on Spark  
|                         | • Native support for transfer learning, Spark DataFrame and ML Pipelines  
<table>
<thead>
<tr>
<th></th>
<th>• Model serving API for model serving/inference pipelines</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Backbends</th>
<th>Spark, TensorFlow, Keras, BigDL, OpenVINO, MKL-DNN, etc.</th>
</tr>
</thead>
</table>

[https://github.com/intel-analytics/analytics-zoo/](https://github.com/intel-analytics/analytics-zoo/)  
[https://analytics-zoo.github.io/](https://analytics-zoo.github.io/)
Analytics Zoo

Build end-to-end deep learning applications for big data
- Distributed *TensorFlow* on Spark
- *Keras*-style APIs (with autograd & transfer learning support)
- *nnframes*: native DL support for Spark DataFrames and ML Pipelines
- Built-in *feature engineering* operations for data preprocessing

Productionize deep learning applications for big data at scale
- POJO *model serving* APIs (w/ OpenVINO support)
- Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
- Built-in deep learning *models* and reference *use cases*
Build end-to-end deep learning applications for big data
• Distributed *TensorFlow* on Spark
• *Keras*-style APIs (with autograd & transfer learning support)
• *nnframes*: native DL support for Spark DataFrames and ML Pipelines
• Built-in *feature engineering* operations for data preprocessing

Productionize deep learning applications for big data at scale
• POJO *model serving* APIs (w/ OpenVINO support)
• Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
• Built-in deep learning *models* and reference *use cases*
Distributed TensorFlow on Spark in Analytics Zoo

1. Data wrangling and analysis using PySpark

```python
from zoo import init_nncontext
from zoo.pipeline.api.net import TFDataset

sc = init_nncontext()

# Each record in the train_rdd consists of a list of NumPy ndarrays
train_rdd = sc.parallelize(file_list)
    .map(lambda x: read_image_and_label(x))
    .map(lambda image_label: decode_to_ndarrays(image_label))

# TFDataset represents a distributed set of elements, 
in which each element contains one or more TensorFlow Tensor objects.
dataset = TFDataset.from_rdd(train_rdd,
    names=['features', 'labels'],
    shapes=[[28, 28, 1], [1]],
    types=[tf.float32, tf.int32],
    batch_size=BATCH_SIZE)
```
2. Deep learning model development using TensorFlow

```python
import tensorflow as tf

slim = tf.contrib.slim

images, labels = dataset.tensors
labels = tf.squeeze(labels)
with slim.arg_scope(lenet.lenet_arg_scope()):
    logits, end_points = lenet.lenet(images, num_classes=10, is_training=True)

loss = tf.reduce_mean(tf.losses.sparse_softmax_cross_entropy(logits=logits, labels=labels))
```
3. Distributed training on Spark and BigDL

```python
from zoo.pipeline.api.net import TFOptimizer
from bigdl.optim.optimizer import MaxIteration, Adam, MaxEpoch, TrainSummary

optimizer = TFOptimizer.from_loss(loss, Adam(1e-3))
optimizer.set_train_summary(TrainSummary("/tmp/az_lenet", "lenet"))
optimizer.optimize(end_trigger=MaxEpoch(5))
```
Analytics Zoo

Build end-to-end deep learning applications for big data
• Distributed TensorFlow on Spark
• Keras-style APIs (with autograd & transfer learning support)
• nnframes: native DL support for Spark DataFrames and ML Pipelines
• Built-in feature engineering operations for data preprocessing

Productionize deep learning applications for big data at scale
• POJO model serving APIs (w/ OpenVINO support)
• Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
• Built-in deep learning models and reference use cases
Keras, Autograd & Transfer Learning APIs

1. Use transfer learning APIs to
   • Load an existing Caffe model
   • Remove last few layers
   • Freeze first few layers
   • Append a few layers

```python
from zoo.pipeline.api.net import *
full_model = Net.load_caffe(def_path, model_path)
# Remove layers after pool5
model = full_model.new_graph(outputs=['pool5'])
# freeze layers from input to res4f inclusive
model.freeze_up_to(['res4f'])
# append a few layers
image = Input(name="input", shape=(3, 224, 224))
resnet = model.to_keras()(image)
resnet50 = Flatten()(resnet)
```

Build Siamese Network Using Transfer Learning
2. **Use Keras-style and autograd APIs to build the Siamese Network**

```python
import zoo.pipeline.api.autograd as A
from zoo.pipeline.api.keras.layers import *
from zoo.pipeline.api.keras.models import *

input = Input(shape=[2, 3, 226, 226])
features = TimeDistributed(layer=resnet50)(input)
f1 = features.index_select(1, 0) # image1
f2 = features.index_select(1, 1) # image2
diff = A.abs(f1 - f2)
fc = Dense(1)(diff)
output = Activation("sigmoid")(fc)
model = Model(input, output)
```
Build end-to-end deep learning applications for big data
- Distributed TensorFlow on Spark
- Keras-style APIs (with autograd & transfer learning support)
- **nnframes**: native DL support for Spark DataFrames and ML Pipelines
- Built-in feature engineering operations for data preprocessing

Productionize deep learning applications for big data at scale
- POJO model serving APIs (w/ OpenVINO support)
- Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
- Built-in deep learning models and reference use cases
nnframes
Native DL support in Spark DataFrames and ML Pipelines

1. Initialize \textit{NNContext} and load images into \textit{DataFrames} using \textit{NNImageReader}

```python
from zoo.common.nncontext import *
from zoo.pipeline.nnframes import *
sc = init_nncontext()
imageDF = NNImageReader.readImages(image_path, sc)
```

2. Process loaded data using \textit{DataFrame} transformations

```python
getName = udf(lambda row: ...)
df = imageDF.withColumn("name", getName(col("image")))
```

3. Processing image using built-in \textit{feature engineering} operations

```python
from zoo.feature.image import *
transformer = ChainedPreprocessing(
    [RowToImageFeature(), ImageChannelNormalize(123.0, 117.0, 104.0),
     ImageMatToTensor(), ImageFeatureToTensor()])
```
4. Define model using Keras-style API

```python
from zoo.pipeline.api.keras.layers import *
from zoo.pipeline.api.keras.models import *
model = Sequential()
    .add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1, 28, 28)))
    .add(MaxPooling2D(pool_size=(2, 2)))
    .add(Flatten()).add(Dense(10, activation='softmax'))
```

5. Train model using Spark ML Pipelines

```python
Estimator = NNEstimater(model, CrossEntropyCriterion(), transformer)
    .setLearningRate(0.003).setBatchSize(40).setMaxEpoch(1)
    .setFeaturesCol("image").setCachingSample(False)
nnModel = estimater.fit(df)
```
Analytics Zoo

Build end-to-end deep learning applications for big data
- Distributed TensorFlow on Spark
- Keras-style APIs (with autograd & transfer learning support)
- nnframes: native DL support for Spark DataFrames and ML Pipelines
- Built-in feature engineering operations for data preprocessing

Productionize deep learning applications for big data at scale
- POJO model serving APIs (w/ OpenVINO support)
- Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
- Built-in deep learning models and reference use cases
Working with Image

1. **Read images into local or distributed ImageSet**
   ```python
   from zoo.common.nncontext import *
   from zoo.feature.image import *
   spark = init_nncontext()
   local_image_set = ImageSet.read(image_path)
   distributed_image_set = ImageSet.read(image_path, spark, 2)
   ```

2. **Image augmentations using built-in ImageProcessing operations**
   ```python
   transformer = ChainedPreprocessing([ImageBytesToMat(),
                                       ImageColorJitter(),
                                       ImageExpand(max_expand_ratio=2.0),
                                       ImageResize(300, 300, -1),
                                       ImageHFlip()])
   new_local_image_set = transformer(local_image_set)
   new_distributed_image_set = transformer(distributed_image_set)
   ```

Image Augmentations Using Built-in Image Transformations (w/ OpenCV on Spark)
Working with Text

1. **Read text into local or distributed TextSet**

   ```python
   from zoo.common.nncontext import *
   from zoo.feature.text import *
   spark = init_nncontext()
   local_text_set = TextSet.read(text_path)
   distributed_text_set = TextSet.read(text_path, spark, 2)
   ```

2. **Build text transformation pipeline using built-in operations**

   ```python
   transformedTextSet = textSet.tokenize()  \   
   .normalize()  \   
   .word2idx()  \   
   .shapeSequence(len)  \   
   .generateSample()  \   
   ```
Analytics Zoo

Build end-to-end deep learning applications for big data
• Distributed TensorFlow on Spark
• Keras-style APIs (with autograd & transfer learning support)
• nnframes: native DL support for Spark DataFrames and ML Pipelines
• Built-in feature engineering operations for data preprocessing

Productionize deep learning applications for big data at scale
• POJO model serving APIs (w/ OpenVINO support)
• Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions
• Built-in deep learning models and reference use cases
import com.intel.analytics.zoo.pipeline.inference.AbstractInferenceModel;

public class TextClassification extends AbstractInferenceModel {
    public RankerInferenceModel(int concurrentNum) {
        super(concurrentNum);
    }
    ...
}

public class ServingExample {
    public static void main(String[] args) throws IOException {
        TextClassification model = new TextClassification();
        
        model.load(modelPath, weightPath);
        
        texts = ...
        List<JTensor> inputs = preprocess(texts);
        for (JTensor input : inputs) {
            List<Float> result = model.predict(input.getData(), input.getShape());
            ...
        }
    }
}
from zoo.common.nncontext import init_nncontext
from zoo.feature.image import ImageSet
from zoo.pipeline.inference import InferenceModel

sc = init_nncontext("OpenVINO Object Detection Inference Example")
images = ImageSet.read(options.img_path, sc,
                        resize_height=600, resize_width=600).get_image().collect()
input_data = np.concatenate([image.reshape((1, 1) + image.shape) for image in images], axis=0)

model = InferenceModel()
model.load_tf(options.model_path, backend="openvino", model_type=options.model_type)
predictions = model.predict(input_data)

# Print the detection result of the first image.
print(predictions[0])

Transparently support OpenVINO in model serving, which deliver a significant boost for inference speed
Model Serving & Inference

Seamless integration in Web Services, Storm, Flink, Kafka, etc. (using POJO local Java APIs)
Build end-to-end deep learning applications for big data

- Distributed TensorFlow on Spark
- Keras-style APIs (with autograd & transfer learning support)
- nnframes: native DL support for Spark DataFrames and ML Pipelines
- Built-in feature engineering operations for data preprocessing

Productionize deep learning applications for big data at scale

- POJO model serving APIs (w/ OpenVINO support)
- Support Web Services, Spark, Storm, Flink, Kafka, etc.

Out-of-the-box solutions

- Built-in deep learning models and reference use cases
Built-in Deep Learning Models

- **Object detection**
  - E.g., SSD, Faster-RCNN, etc.

- **Image classification**
  - E.g., VGG, Inception, ResNet, MobileNet, etc.

- **Text classification**
  - Text classifier (using CNN, LSTM, etc.)

- **Recommendation**
  - E.g., Neural Collaborative Filtering, Wide and Deep Learning, etc.

- **Anomaly detection**
  - Unsupervised time series anomaly detection using LSTM

- **Sequence-to-sequence**
Object Detection API

1. Load pretrained model in *Detection Model Zoo*

```python
from zoo.common.nncontext import *
from zoo.models.image.objectdetection import *
spark = init_nncontext()
model = ObjectDetector.load_model(model_path)
```

2. Off-the-shell inference using the loaded model

```python
image_set = ImageSet.read(img_path, spark)
output = model.predict_image_set(image_set)
```

3. Visualize the results using utility methods

```python
config = model.get_config()
visualizer = Visualizer(config.label_map(), encoding="jpg")
visualized = visualizer(output).get_image(to_chw=False).collect()
```

[Off-the-shell Inference Using Analytics Zoo Object Detection API](https://github.com/intel-analytics/analytics-zoo/tree/master/pyzoo/zoo/examples/objectdetection)
Sequence-to-Sequence API

\[
\text{encoder = RNNEncoder.initialize(rnn_type, nlayers, hidden_size, embedding)}
\]
\[
\text{encoder = RNNDecoder.initialize(rnn_type, nlayers, hidden_size, embedding)}
\]
\[
\text{seq2seq = Seq2seq(encoder, decoder)}
\]
Reference Use Cases

• **Anomaly Detection**
  • Using LSTM network to detect anomalies in time series data

• **Fraud Detection**
  • Using feed-forward neural network to detect frauds in credit card transaction data

• **Recommendation**
  • Use Analytics Zoo Recommendation API (i.e., Neural Collaborative Filtering, Wide and Deep Learning) for recommendations on data with explicit feedback.

• **Sentiment Analysis**
  • Sentiment analysis using neural network models (e.g. CNN, LSTM, GRU, Bi-LSTM)

• **Variational Autoencoder (VAE)**
  • Use VAE to generate faces and digital numbers

• **Web services**
  • Use Analytics Zoo model serving APIs for model inference in web servers

https://github.com/intel-analytics/analytics-zoo/tree/master/apps
Analytics Zoo Examples

Dogs vs. cats, object detections, Distributed TF
Dogs vs. Cats

Notebook:
Object Detection API

Notebook:

Image Classification & Fine-Tuning Using TFNet

Notebook:

Distributed TensorFlow Training on Spark

https://github.com/intel-analytics/analytics-zoo/blob/master/pyzoo/zoo/examples/tensorflow/distributed_training/train_lenet.py

https://github.com/intel-analytics/analytics-zoo/blob/master/pyzoo/zoo/examples/tensorflow/distributed_training/train_mnist_keras.py
Break
Distributed Training In BigDL

Data parallel training
Parameter synchronization
Scaling and Convergence
Task scheduling

“BigDL: A Distributed Deep Learning Framework for Big Data”
Apache Spark

[Diagram showing a single master (driver) and multiple workers, with Spark tasks indicated]
Spark compute model

- Data parallel
- Functional, coarse-grained operators
  - Immutable RDDs
  - Applying the same operation (e.g., `map`, `filter`, etc.) to all data items

Source: “Parallel programming with Spark”, Matei Zaharia, AMPCamp 3
Prepare training data as an RDD of \textit{Samples}
Construct an RDD of \textit{models} (each being a replica of the original model)

\begin{verbatim}
for (i <- 1 to N) {
  //"model forward-backward" job
  for each task in the Spark job:
    read the latest \textit{weights}
    get a random \textit{batch} of data from local \textit{Sample} partition
    compute errors (forward on local model replica)
    compute \textit{gradients} (backward on local model replica)

  //"parameter synchronization" job
  aggregate (sum) all the \textit{gradients}
  update the \textit{weights} per specified optimization method
}
\end{verbatim}
Data Parallel Training

Task 1: Sample RDD

Task 2: Model RDD

Task n: zip Sample and model RDDs, and compute gradient on co-located Sample and model partitions

“Model Forward-Backward” Job
Parameter Synchronization

\[
\sum \text{gradient } i \\
\sum \text{weight } i \\
\text{update}
\]

"Parameter Synchronization" Job
For each task $n$ in the "parameter synchronization" job {
    shuffle the $n^{th}$ partition of all gradients to this task
    aggregate (sum) the gradients
    updates the $n^{th}$ partition of the weights
    broadcast the $n^{th}$ partition of the updated weights
}
Training Scalability

Throughput of ImageNet Inception v1 training (w/ BigDL 0.3.0 and dual-socket Intel Broadwell 2.1 GHz); the throughput scales almost linear up to 128 nodes (and continue to scale reasonably up to 256 nodes).

Increased Mini-Batch Size

• Distributed synchronous mini-batch SGD
  • Increased mini-batch size
    \[ \text{total\_batch\_size} = \text{batch\_size\_per\_worker} \times \text{num\_of\_workers} \]
  • Can lead to loss in test accuracy

• State-of-art method for scaling mini-batch size*
  • Linear scaling rule
  • Warm-up strategy
  • Layer-wise adaptive rate scaling
  • Adding batch normalization

Training Convergence: Inception v1

**Strategies**
- Warm-up
- Linear scaling
- Gradient clipping
- TODO: adding batch normalization

Source: Very large-scale distributed deep learning with BigDL, Jason Dai and Ding Ding. O'Reilly AI Conference 2017
Training Convergence: SSD

Source: Very large-scale distributed deep learning with BigDL, Jason Dai and Ding Ding. O'Reilly AI Conference 2017

Strategies
- Warm-up
- Linear scaling
- Gradient clipping
## Difference vs. Classical PS Architecture

<table>
<thead>
<tr>
<th>Classical PS architecture</th>
<th>BigDL implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Multiple long-running, potentially stateful tasks</td>
<td>• Run a series of short-lived Spark jobs (e.g., two jobs per mini-batch)</td>
</tr>
<tr>
<td>• Interact with each other (in a blocking fashion for synchronization)</td>
<td>• Each task in the job is stateless and non-blocking</td>
</tr>
<tr>
<td>• Require fine-grained data access and in-place data mutation</td>
<td>• Automatically adapt to the dynamic resource changes (e.g., preemption, failures, resource sharing, etc.)</td>
</tr>
<tr>
<td>• Not directly supported by existing big data systems</td>
<td>• Built on top of existing primitives in Spark (e.g., shuffle, broadcast, and in-memory data persistence)</td>
</tr>
</tbody>
</table>
Task Scheduling Overheads

Sparks overheads (task scheduling & task dispatch) as a fraction of average compute time for Inception v1 training

BigDL implementations
- Run a single, multi-threaded task on each worker
- Achieve high scalability on large clusters (e.g., up to 256 servers)

Reducing Scheduling Overheads Using Drizzle

Scaling to even larger (>500) workers

- Iterative model training
  - Same operations run repeatedly
- Drizzle
  - A low latency execution engine for Spark
  - Group scheduling for multiple iterations of computations at once

Advanced Analytics Zoo Applications

Text classification, movie recommendations, Q&A ranker
Text Classification

Movie Recommendations

Notebook:
Real-World Applications

Object detection and image feature extraction at JD.com
Produce defect detection using distributed TF on Spark in Midea
NLP based customer service chatbot for Microsoft Azure
Image similarity based house recommendation for MLSlisting
Transfer learning based image classifications for World Bank
LSTM-Based time series anomaly detection for Baosight
Fraud detection for payment transactions for UnionPay
Object Detection and Image Feature Extraction at JD.com
Applications

Large-scale image feature extraction
• Object detect (remove background, optional)
• Feature extraction

Application
• Similar image search
• Image Deduplication
  • Competitive price monitoring
  • IP (image copyright) protection system

Source: “Bringing deep learning into big data analytics using BigDL”, Xianyan Jia and Zhenhua Wang, Strata Data Conference Singapore 2017
Similar Image Search

Source: “Bringing deep learning into big data analytics using BigDL”, Xianyan Jia and Zhenhua Wang, Strata Data Conference Singapore 2017
Challenges of Productionizing Large-Scale Deep Learning Solutions

**Productionizing large-scale deep learning solutions is challenging**

- Very complex and error-prone in managing large-scale distributed systems
  - E.g., resource management and allocation, data partitioning, task balance, fault tolerance, model deployment, etc.

- Low end-to-end performance in GPU solutions
  - E.g., reading images out from HBase takes about half of the total time

- Very inefficient to develop the end-to-end processing pipeline
  - E.g., image pre-processing on HBase can be very complex
Production Deployment with Analytics Zoo for Spark and BigDL

- Reuse existing Hadoop/Spark clusters for deep learning with no changes (image search, IP protection, etc.)
- Efficiently scale out on Spark with superior performance (3.83x speed-up vs. GPU servers) as benchmarked by JD

http://mp.weixin.qq.com/s/xUckzbHK4K06-v5qUsaNQQ
Produce Defect Detection using Distributed TF on Spark in Midea

Produce Defect Detection using Distributed TF on Spark in Midea
NLP Based Customer Service Chatbot for Microsoft Azure

Image Similarity Based House Recommendation for MLSlistings

MLSlistings built image-similarity based house recommendations using BigDL on Microsoft Azure

Image Similarity Based House Recommendation for MLSlistings

- RDD of house photos
- Image pre-processing
- Pre-trained VGG16 model (to extract features)
- Three pre-trained Inception v1 models (fine-tuned as classifiers)
- Image features
- Store image tags and feature in table storage
- Tags (is_exterior, style, floors) of images
- Image embedding 25088 floats

Image Processing

Tags (is_exterior, style, floors) of images
Image Similarity Based House Recommendation for MLSTlistings

Notebook:
Transfer Learning Based Image Classifications for World Bank

Classifying Real Food Images is not a Cat vs. Dog Problem

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018
Project Layout

Phase 1:
- Image preprocessing (eliminate poor quality images and invalid images)
- Classify images (by food type) to validate existing labels

Phase 2:
- Identify texts in the image and make bounding box around them
- Text recognition (words/sentences in the image text)
- Determine whether text contains PII (personal identifiable information)
- Blur areas with PII text

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018
Code – Phase 1

Fine-tuning Training

```java
# get model
pretrained_model_path = path.join(MODEL_ROOT, "bigdl_inception_v1_imagenet_0_4_0.model")

# create a new model by remove layers after pool5/drop_7x7_s1
model = full_model.new_graph(["pool5/drop_7x7_s1"])

# inputNode = Input(name="input", shape=(3, 224, 224))
# inception = model.to_keras()(inputNode)
# flatten = Flatten()(inception)
# logits = Dense(n_classes)(flatten)

createZooKerasInput()
createZooKerasFlatten()
createZooKerasDense()
createZooKerasModel()

Command took 4.74 seconds -- by Jiao.Wang@intel.com at 6/2/2018, 8:03:46 PM on 20-node-cluster
```

Prediction and Evaluation

```java
# predict
predict_model = trained_model.setBatchSize(batch_size)
predictionDF = predict_model.transform(test_image)
predictionDF.cache()

/// Measure Test Accuracy w/Test Set
#

evaluator = MulticlassClassificationEvaluator(labelCol="label",
                                                predictionCol="prediction",
                                                metricName="accuracy")

accuracy = evaluator.evaluate(predictionDF)

# expected error should be less than 10%
print("Accuracy = %g % accuracy")
predictionDF.unpersist()
```

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018
Result – Phase 1

- Fine tune with Inception v1 on a full dataset
- Dataset: 994325 images, 69 categories

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cores</th>
<th>Batch Size</th>
<th>Epochs</th>
<th>Training Time (sec)</th>
<th>Throughput (images/sec)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
<td>1200</td>
<td>12</td>
<td>61125</td>
<td>170</td>
<td>81.7</td>
</tr>
</tbody>
</table>

* This model training was performed using multinode cluster on AWS R4.8xlarge instance with 20 nodes

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018
Next Steps – Phase 2

• Image Quality Preprocessing
  • Filter with print text only
  • Rescaling, Binarisation, Noise Removal, Rotation / Deskewing (OpenCV, Python, etc.)

• Detect text and bounding box circle text
• Recognize text
• Determine whether text contains PII (personal identifiable information)
  • Recognize PII with leading words
• Blur areas with PII text
  • Image tools

Source: Using Crowdsourced Images to Create Image Recognition Models with Analytics Zoo using BigDL, Maurice Nsabimana and Jiao Wang, Spark Summit 2018
LSTM-Based Time Series Anomaly Detection for Baosight

LSTM-Based Time Series Anomaly Detection for Baosight

Notebook:
Fraud Detection for Payment Transactions for UnionPay

Training Data

- Normal
  - Pre-processing
  - Spark Pipeline
  - Feature Engineering
    - Sampled partition
    - All features

- Fraud
  - Pre-processing
  - Spark Pipeline
  - Feature Engineering
    - Sampled partition
  - Sampled partition

Spark Pipeline

- Feature Selection
  - Selected features
  - Model Training
    - Model candidate
  - Model Evaluation & Fine Tune
    - Model
    - Spark Pipeline
      - Feature Engineering
      - Feature Selection
      - Model Ensemble

Pre-processing

- Predictions

Post-processing

- Test Data

Hive Table

Spark DataFrame

Neural Network Model Using BigDL

https://mp.weixin.qq.com/s?__biz=MzI3NDAwNDUwNg==&mid=2648307335&idx=1&sn=8eb9f63eaf2e40e24a90601b9cc03d1f
Fraud Detection for Payment Transactions for UnionPay

Notebook:

Unified Analytics + AI Platform
Distributed TensorFlow, Keras and BigDL on Apache Spark
https://github.com/intel-analytics/analytics-zoo
**LEGAL DISCLAIMERS**

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit [http://www.intel.com/performance](http://www.intel.com/performance).

Intel, the Intel logo, Xeon, Xeon phi, Lake Crest, etc. are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2019 Intel Corporation