Performance Challenges of DeepLearning

- Compute intensive process

Solutions

- Move from CPUs to GPUs
- Parallelism
- Shard the dataset
- Train on Multiple Cores
- Train on Multiple Machines

Deep Learning Review

- Simple Neural Network
- Classify Flowers by measurements

The Data

```
5.1,3.5,1.4,0.2,0
4.9,3.0,1.4,0.2,0
4.7,3.2,1.3,0.2,0
4.6,3.1,1.5,0.2,0
```

skymind

The Code

- https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/dataexamples/CSVExample.java

Code Overview

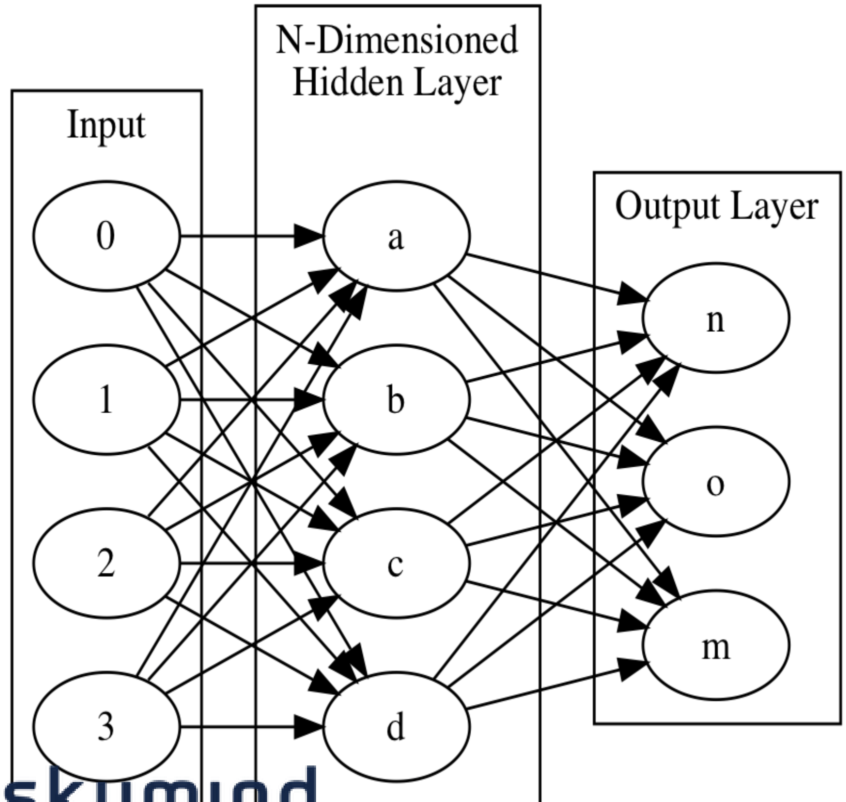
- Read the Data
- Build a model
- Train the model

Performance Data Iris Model

- read 150 lines 3KB
- Train the model 39 parameters

skymind

Picture of Model Graph



VGG16

- Read 14,197,122 images
- Augment with image transformations
- Manage 13,8357,544 parameters
- 21 Layers
- Train for 72 epochs

Picture of VGG

Options for Increasing Performance

skymind

Options for executing code

- CPU
- GPU
- Multi GPU
- Spark
- SPark + GPU

Architecture Details

- libnd4j
  - The C++ engine that powers ND4J
- ND4J
  - n-dimensional arrays for java
- Datavec
  - ETL
- DeepLEarning4j
  - Build, train and deploy models

skymind

ND4J

- Where the math happens
- ND4J-backend
- Determines how execution happens

skymind

Execute on CPU

```
<Your pom.xml file>
<properties>
<nd4j.backend>nd4j-native-platform</nd4j.backend>
```

Execute on GPU

```
<Your pom.xml>
<properties>
<nd4j.backend>nd4j-cuda-8.0-platform</<nd4j.backend>
```

skymind

Troubleshooting

- Failure to use multiple GPU's

```
//Add this to your main
CudaEnvironment.getInstance().getConfiguration()
.allowMultiGPU(true);
```

Multi-GPU Data Parallelism

- ParallelWrapper
  - Model Config remains the same
  - ND4J-Backend set to GPU

    ParallelWrapper wrapper = new ParallelWrapper.Builder(YourExistingModel) .prefetchBuffer(24) .workers(4) .averagingFrequency(1) .reportScoreAfterAveraging(true) .useLegacyAveraging(false) .build();

skymind

ParallelWrapper Internals

- Model will be duplicated
- Each worker trains model
- Models averaged at averagingFrequency(X)
- Training Continues

Distributed Training in Spark

- Documentation
  - https://deeplearning4j.org/spark
- Examples
  - https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-spark-examples

skymind

Training on Spark

- SparkDl4jMultiLayer vs MultiLayerNetwork
- SparkComputationGraph vs ComputationGraph

Non-Distributed Training overview

- Process minibatch
  - Calculate Loss Function
  - Calculate direction of less error
  - Update weights in that direction

Non-Distributed Training Challenges

- Number of parameters may be large
- CPU or preferably GPU intensive
- Update large multi-dimensional matrix of numeric values

Data Parallelism vs Model Parallelism

- Data Parallelism

Different machines have a complete copy of the model; each machine simply gets a different portion of the data, and results from each are combined.

- Model Parallelism

different machines in the distributed system are responsible for the computations in different parts of a single network - for example, each layer in the neural network may be assigned to a different machine.

skymind

Data Parallelism

- Our current approach

Data Parallel Theoretical options

- Parameter averaging vs. update (gradient)-based approaches
- Synchronous vs. asynchronous methods
- Centralized vs. distributed synchronization

skymind

DL4J Current approach

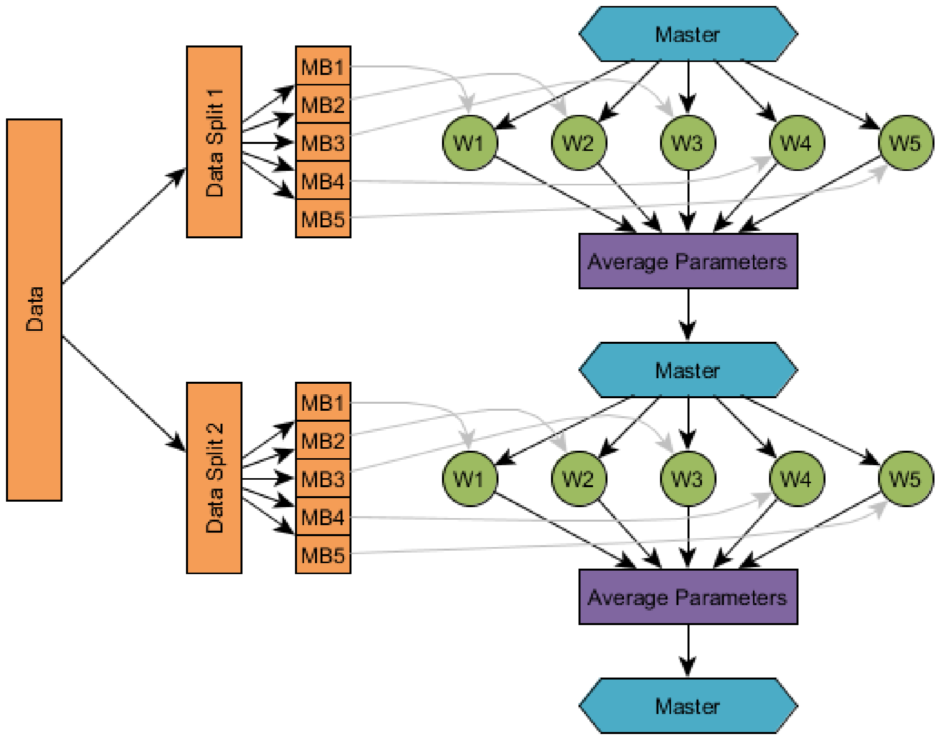
- Synchronous Parameter Averaging

Parameter Averaging Overview

1. Initialize the network parameters randomly based on the model configuration
2. Distribute a copy of the current parameters to each worker
3. Train each worker on a subset of the data
4. Set the global parameters to the average the parameters from each worker
5. While there is more data to process, go to step 2

Distributed Training Implementation

- Workers process minibatch
- Calculate Loss Function
- Calculate Gradient update after each minibatch
- Submit to Parameter server
- Parameter Server averages weights from workers
- Ships averaged weights to workers


Data
Data Split 1
MB1
MB2
MB3
MB4
MB5
Master
W1
W2
W3
W4
W5
Average Parameters
Master
Data Split 2
MB1
MB2
MB3
MB4
MB5
W1
W2
W3
W4
W5
Average Parameters
Master

Parameter Averaging Considerations

- Distributed Systems 101
- Synchronization is a challenge to performance
- Sharing is a challenge to performance

Tuning Parameter Averaging

- Setting Averaging Period
- Increase and workers may diverge
- Decrease and cost of Synching increases
- Advice
  - Greater than 1
  - Less than 20
  - Between 5-10

skymind

Batch Prefetch

Spark workers are capable of asynchorously prefetching a number of minibatches (DataSet objects), to avoid waiting for the data to be loaded.

- Values
    - 0 disables prefetching.
    - 2 is often a sensible default.
    - Too Large and Memory is wasted with no additional benefit

Updater State

- momentum, RMSProp and AdaGrad have internal state
- saveUpdater == true
    - updater state (at each worker) will be averaged and returned to the master along with the parameters
    - Extra time and Network Traffic, may improve performance

skymind

More Options

- See https://deeplearning4j.org/spark

DL4J Spark Examples

- https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-spark-examples

Minimal Example

```
JavaSparkContent sc = ...;
JavaRDD<DataSet> trainingData = ...;
MultiLayerConfiguration networkConfig = ...;

//Create the TrainingMaster instance
int examplesPerDataSetObject = 1;
TrainingMaster trainingMaster = new ParameterAveragingTrainingMaster.Builder(exampl
    .(other configuration options)
    .build();

//Create the SparkDl4jMultiLayer instance
SparkDl4jMultiLayer sparkNetwork = new SparkDl4jMultiLayer(sc, networkConfig, train

//Fit the network using the training data:
sparkNetwork.fit(trainingData);
```

skymind

How to Participate and Contribute

- Chat with us on Gitter
    - https://gitter.im/deeplearning4j/deeplearning4j
- Contribute
    - https://github.com/deeplearning4j/

skymind

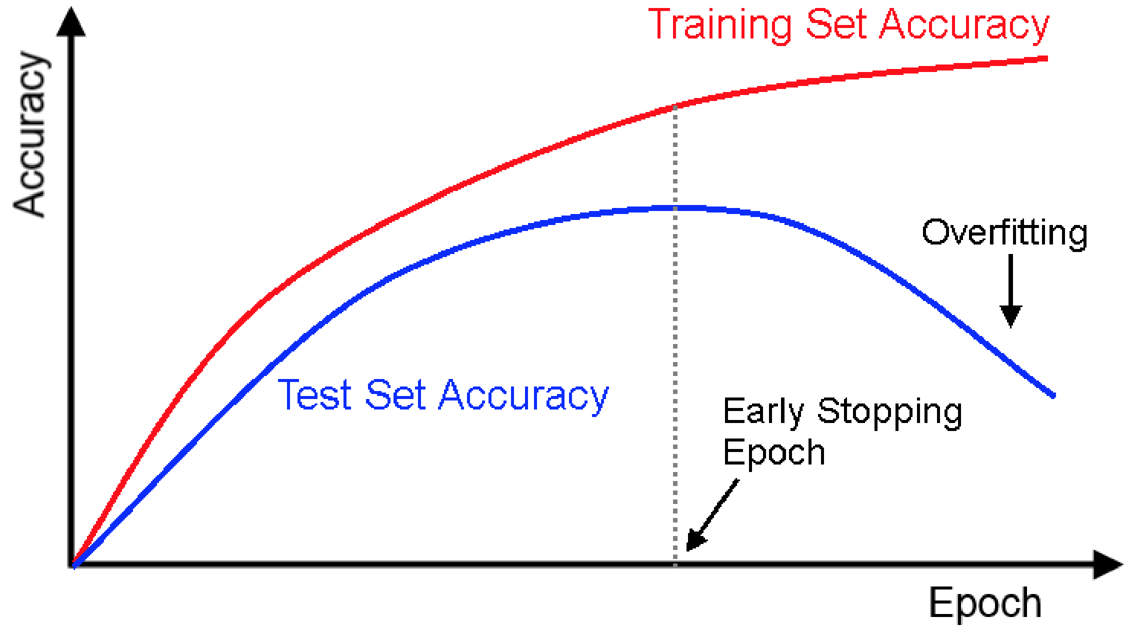
Interesting Challenges

- GPU aware Yarn
    - https://issues.apache.org/jira/browse/YARN-5517
- Parallelism in DeepLearning
    - https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_netwo

Early Stopping

- Allows Training to stop before overfitting occurs

Early Stopping Image



skymind

ParallelWrapper and Early Stopping

- What is Early Stopping?
  - Saves Model that performs best on Test Data
  - Saves Model before training overfits training data

skymind

Training on Single GPU or CPU

- Use EarlyStoppingTrainer

Training on Multi-GPUs

- Use EarlyStoppingParallelTrainer

Training on Spark

- Use SparkEarlyStoppingTrainer

skymind

Examples

Main Examples Repo

https://github.com/deeplearning4j/dl4j-examples

Quick Start Guide

- https://deeplearning4j.org/quickstart

Spark Examples

- https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-spark-examples
- Includes
  - EMR
  - Distributed Spark MLP example
  - LSTM on Spark Example
  - Statistics Gathering Example

skymind

GPU Examples

- https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-cuda-specific-examples

Material included for home study

The following materials are included for your use, not inteded to be covered during presentation

DataVec: Spark Transform Process

DataVec overview

- Goal of DataVec
- Data => INDArray

DataVec Spark Based Operations

- Joins
- Column Transformations
- Data Analysis

Spark Analyze

- Tool to gather statistics across a data set

Spark Analyze Example

- The Data: classic dataset of Iris flower characteristics

```
5.1,3.5,1.4,0.2,0
4.9,3.0,1.4,0.2,0
4.7,3.2,1.3,0.2,0
4.6,3.1,1.5,0.2,0
5.0,3.6,1.4,0.2,0
5.4,3.9,1.7,0.4,0
```

skymind

Spark Analyze Example Continued...

- Define a Schema

```
Schema schema = new Schema.Builder()
.addColumnsDouble("Sepal length", "Sepal width",
"Petal length", "Petal width")
.addColumnInteger("Species")
.build();
```

Spark Analyze Example Continued...

- Create Spark Conf

```
SparkConf conf = new SparkConf();
conf.setMaster("local[*]");
conf.setAppName("DataVec Example");
JavaSparkContext sc = new JavaSparkContext(conf);
```

Spark Analyze Example Continued...

- Read Data

```
String directory = new ClassPathResource("IrisData/iris.txt")
.getFile().getParent();
//Normally just define your directory
//like "file:/..." or "hdfs:/..."
JavaRDD<String> stringData = sc.textFile(directory);
```

skymind

Spark Analyze Example Continued…

- Parse Data

```
//We first need to parse this comma-delimited (CSV) format;
//we can do this using CSVRecordReader:
RecordReader rr = new CSVRecordReader();
JavaRDD<List<Writable>> parsedInputData =
stringData.map(new StringToWritablesFunction(rr));
```

Spark Analyze Example Continued…

- Analyze Data

```
int maxHistogramBuckets = 10;
DataAnalysis dataAnalysis =
AnalyzeSpark.analyze(schema,
parsedInputData, maxHistogramBuckets);

System.out.println(dataAnalysis);
```

skymind

Spark Analyze Output

- min,max,mean,StDev,Variance,num0,
- numNegative,numPositive,numMin,numMax,Total
- See example IrisAnalysis.java

skymind

Spark Analyze options

- Per Column Analysis

```
//We can get statistics on a per-column basis:
DoubleAnalysis da = (DoubleAnalysis)dataAnalysis.getColumnAnalysis("Sepal
double minValue = da.getMin();
double maxValue = da.getMax();
double mean = da.getMean();
```

skymind

Spark Analyze HTML output

- To Generate HTML Output

```
HtmlAnalysis.createHtmlAnalysisFile(dataAnalysis,
new File("DataVecIrisAnalysis.html"));

//To write to HDFS instead:
//String htmlAnalysisFileContents =
//HtmlAnalysis.createHtmlAnalysisString(dataAnalysis);
//SparkUtils.writeStringToFile("hdfs://your/hdfs/path/here",
//htmlAnalysisFileContents,sc);
```

skymind

DataVec Spark Transform Process

- Joining
- Transformation
- Schema alteration

Video

- https://www.youtube.com/watch?v=MLEMw2NxjxE

skymind

Code Examples

- https://github.com/deeplearning4j/dl4j-examples/tree/master/datavec-examples
- http://github.com/SkymindIO/screencasts/tree/master/datavec_spark_transform

skymind

Basic Example CSV Data

- Storm Reports Data

```
161006-1655,UNK,2 SE BARTLETT,LABETTE,KS,37.03,-95.19,
TRAINED SPOTTER REPORTS TORNADO ON THE GROUND. (ICT),TOR

//Fields are
//datetime,severity,location,county,state,lat,lon,comment,type
```

skymind

Define Schema as read

```
Schema inputDataSchema = new Schema.Builder()
.addColumnsString("datetime","severity","location","county","state")
.addColumnsDouble("lat","lon")
.addColumnsString("comment")
.addColumnCategorical("type","TOR","WIND","HAIL")
.build();
```

Define Transform process

```
TransformProcess tp = new TransformProcess.Builder(inputDataSchema)
.removeColumns
("datetime","severity","location","county","state","comment")
.categoricalToInteger("type")
.build();
// keep lat/lon convert type to 0,1,2
```

Create Spark Conf

```
SparkConf sparkConf = new SparkConf();
sparkConf.setMaster("local[*]");
sparkConf.setAppName("Storm Reports Record Reader Transform");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
```

skymind

read the data file

```
JavaRDD<String> lines = sc.textFile(inputPath);
```

skymind

Convert to Writable

```
JavaRDD<List<Writable>> stormReports =
lines.map(new StringToWritablesFunction(new CSVRecordReader()));
```

Run our transform process

```
JavaRDD<List<Writable>> processed =
SparkTransformExecutor.execute(stormReports,tp);
```

Convert Writable Back to String for Export

```
JavaRDD<String> toSave=
processed.map(new WritablesToStringFunction(","));
```

skymind

Write to File

```
toSave.saveAsTextFile(outputPath);
```