DL4J COMPONENTS

- Open Source DeepLearning Library
- CPU & GPU support
- Hadoop-Yarn & Spark integration
- Future additions
DL4J COMPONENTS

- Open Source Deep Learning Library

What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support
- Hadoop-Yarn & Spark integration
- Future additions
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn & Spark integration
  What components are interfaced? (Section 2)

- Future additions
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)

- Future additions
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)

  How are the algorithms distributed? (Section 3)

- Future additions
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is Deep Learning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)

- Future additions

Section 4 / The End.
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)
- CPU & GPU support
- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)
- Future additions
  Section 4 / The End.
Let's first look at what actually Deep Learning is.

Generally: Deep Learning might be only an alternative term for greater sized artificial neural networks?

Bigger Datasets are also required (scaling/performance issues?)
DL4J offers the following Deep Neural Networks:

- Restricted Boltzmann Machines
- Convolutional Nets (Images)
- Recurrent nets/LSTMs
- Recursive Autoencoders
- Deep Belief Networks
- Recursive Neural Tensor (~Vector/Array) Networks
- Stacked Denoising Autoencoders

Data frame (/set) readers included for: MNIST, Labeled Faces in the Wild (LFW), IRIS

And DL4J also supports pipelining / stacking of layers.
DEEP ARTIFICIAL NEURAL NETWORKS (DNNS)

Supervised Learning
Approximate pre-defined lables by output of network (e.g. classify faces as 1 and everything else as 0); can use output from unsupervised learning (e.g. DBNs) for weight initialization

Recursive Neural Networks
Apply the same layer multiple times

Convolutional Neural Networks (CNNs)
Tries to copy natural visual system (see next slides)

s = \text{score} \cdot p \quad (9)
p = f(W[c_1; c_2] + b)

Restricted Boltzmann Machine
Autoencoder
Deep Boltzmann Machine
Stacked Autoencoder
Denoising Autoencoder
Sparse Autoencoder

Deep Belief Networks (DBNs)
Aims at a good stochastic reconstruction of the input

Convolutional Deep Belief Networks (CDBNs)

Semi-Supervised Learning
Combine both.

Reinforcement Learning
Define a reward function and try to gain as much reward as possible as a result of the chosen actions (could be considered as somewhat supervised);
also here some work is currently in progress DNN implementing this type of learning (some of those above can be adapted to do so)

More Info & Sources:
https://www.youtube.com/watch?v=oRxgRHJztPI
https://www.youtube.com/watch?v=IekCh_i32IE
Recurrent connections in Hippocampus enable sequential memory and different types of time depended learning.

Fig. 3. The numbers of connections from three different sources onto each CA3 cell from three different sources in the rat.

MOTIVATION: RECURRENT NEURAL NETWORKS
USES FOR RECURSIVE/RECURRENT NEURAL NETWORKS

Figure 1. Illustration of our recursive neural network architecture which parses images and natural language sentences. Segment features and word indices (orange) are first mapped into semantic feature space (blue) and then recursively merged by the same neural network until they represent the entire image or sentence. Both mappings and merging are learned.

**Left:** Socher et al. (2011) - Parsing Natural Scenes and Natural Language with Recursive Neural Networks (Google)

**Top Middle:** Hannun (2014) - Deep Speech: Scaling up end-to-end speech recognition (Baidu)

**Rest:** Karpathy (2015) - The Unreasonable Effectiveness of Recurrent Neural Networks [link]

**Further Readings:** Gillick (2016) - Multilingual Language Processing From Bytes; Sak (2015) - Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition; Marek et al (2015) - Large-Scale Language Classification (200 Languages, Twitter)
// Get a DataSetIterator that handles vectorization of text into something we can use to train
// our GravesLSTM network.
CharArrayIterator it = new CharArrayIterator(new StringReader(example));
Iter iter = new DefaultCharIterator(it);

// Set up network configuration:
MultilayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .optimizationAlgorithm(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT).iterations(1)
    .learningRate(0.1)
    .rmsDecay(0.95)
    .seed(12345)
    .regularization(true)
    .l2(0.001)
    .weightInit(WeightInit.XAVIER)
    .updater(Updater.RMSPROP)
    .list()
    .layer(0, new GravesLSTM.Builder().nIn(iter.inputColumns()).nOut(1) + lstmLayers).activation("tanh").build())
    .layer(1, new GravesLSTM.Builder().nIn(lstmLayers).nOut(lstmLayers) + lstmLayers).activation("tanh").build())
    .layer(2, new RnnOutputLayer.Builder(LossFunction.MCXENT).activation("softmax") + // MCXENT + softmax for classification
    .nIn(lstmLayers).nOut(nOut).build())
    .backpropType(BackpropType.TruncatedBPTT).tBPTTForwardLength(tBpttLength).tBPTTBackwardLength(tBpttLength)
    .pretrain(false).backprop(true)
    .build();

MultilayerNetwork net = new MultilayerNetwork(conf);
net.init();
net.setListeners(new ScoreIterationListener(1));

// Do training, and then generate and print samples from network
int miniBatchNumber = 0;
for( int i=0; i<numEpochs; i++ ){
    while(iter.hasNext()){
        DataSet ds = iter.next();
        net.fit(ds);
        if( ++miniBatchNumber % generateSamplesEveryMMiniBatches == 0){
            System.out.println("--------------");
            System.out.println("Completed " + miniBatchNumber + " minibatches of size " + miniBatchSize + "\" + exampleLen
            System.out.println("Sampling characters from network given initialization \" + (generationInitialization == n
            String[] samples = sampleCharactersFromNetwork(generationInitialization, net, iter, rng, nCharactersToSample, nSamp
            for( int j=0; j<samples.length; j++ ){
                System.out.println("----- Sample " + j + " ----- ");
                System.out.println(samples[j]);
                System.out.println();
            }
        }
    }
}
MOTIVATION: SPARSITY & EXTRACTION OF STIMULY STATISTICS

- High information content (direction of high variance extracted by e.g. Denoising AE / )
- Low overlap of feature maps, so more feat can be extracted
Example: Unsupervised Learning in DL4J

```java
log.info("Build model....");
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
    .gradientNormalizationThreshold(1.0)
    .iterations(iterations)
    .momentum(0.5)
    .momentumAfter(Collections.singletonMap(3, 0.9))
    .optimizationAlgorithm(OptimizationAlgorithm.CONJUGATE_GRADIENT)
    .list(4)
    .layer(0, new AutoEncoder.Builder().nIn(numRows * numColumns).nOut(500)
      .weightInit(WeightInit.XAVIER).lossFunction(LossFunction.RMSE_XENT)
      .corruptionLevel(0.3)
      .build())
    .layer(1, new AutoEncoder.Builder().nIn(500).nOut(250)
      .weightInit(WeightInit.XAVIER).lossFunction(LossFunction.RMSE_XENT)
      .corruptionLevel(0.3)
      .build())
    .layer(2, new AutoEncoder.Builder().nIn(250).nOut(200)
      .weightInit(WeightInit.XAVIER).lossFunction(LossFunction.RMSE_XENT)
      .corruptionLevel(0.3)
      .build())
    .layer(3, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELYOOD).activation("softmax")
      .nIn(200).nOut(outputNum).build())
    .pretrain(true).backprop(false)
    .build();

MultilayerNetwork model = new MultilayerNetwork(conf);
model.init();
model.setListeners(Collections.singletonList(new ScoreIterationListener(listenerFreq)));
log.info("Train model....");
```
The (human) visual system is hierarchically organized (each area thereof consists of multiple neuron layers [I-VI] with different types of connectivity.

**Top:** Scannel (1993) - the connectional organization of the cat visual cortex

**Bottom Right:** Fellemann & Van Essen (1991) - Distributed Hierarchical Processing in the Primate Cerebral Cortex
The (human) visual system is hierarchically organized (each area thereof consists of multiple neuron layers [I-VI] with different types of connectivity.)

**Top:** Scannel (1993) - the connectional organization of the cat visual cortex

**Bottom Right:** Fellemann & Van Essen (1991) - Distributed Hierarchical Processing in the Primate Cerebral Cortex
Hierarchical Processing for Extraction of different features (= Feature Maps) in the Visual System

See next slide for how CNN do this ...
**REAL USES FOR CONVOLUTIONAL NEURAL NETWORKS**

*Left:* Socher et al. (2011) - Parsing Natural Scenes and Natural Language with Recursive Neural Networks (Google)

*Right:* Szegedy (2014) - Going Deeper with Convolutions (GoogLeNet/Inception for Image Classification & Detection)

Figure 3: GoogLeNet network with all the bells and whistles
EXAMPLE: CONVOLUTIONAL NEURAL NETWORKS IN DL4J

```java
log.Info("Build model...");
MultiLayerConfiguration.Builder builder = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(iterations)
    .regularization(true).l2(0.0005)
    .learningRate(0.1)
    .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
    . updater(Updater.ADAGRAD)
    .list()
    .layer(0, new ConvolutionLayer.Builder(5, 5)
        .nIn(nChannels)
        .stride(1, 1)
        .nOut(28)
        .weightInit(WeightInit.XAVIER)
        .activation("relu")
        .build())
    .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX, new int[][, 2, 2])
        . build())
    .layer(2, new ConvolutionLayer.Builder(5, 5)
        .nIn(28)
        .nOut(50)
        .stride(2, 2)
        .weightInit(WeightInit.XAVIER)
        .activation("relu")
        . build())
    .layer(3, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX, new int[][, 2, 2])
        . build())
    .layer(4, new DenseLayer.Builder().activation("relu")
        .weightInit(WeightInit.XAVIER)
        .nOut(280).build())
    .layer(5, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
        .nOut(outputNum)
        .weightInit(WeightInit.XAVIER)
        .activation("softmax")
        .build())
    .backprop(true).pretrain(false);
new ConvolutionLayerSetup(builder, 28, 28, 1);
```
COMBINATION OF DIFFERENT DNNs

Can be done by stacking of different layer types

Top: Vinyals (2015) - Show and Tell: A Neural Image Caption Generator
Right: Sak (2015) – Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks
Example: Chaining Neural Networks in DL4J

Recurrent Network with Skip Connections

```java
ComputationGraphConfiguration conf = new NeuralNetConfiguration.Builder()
    .learningRate(0.01)
    .graphBuilder()
    .addInputs("input") // can use any label for this
    .addLayer("L1", new GravesLSTM.Builder().nIn(5).nOut(5).build(), "input")
    .addLayer("L2", new RnnOutputLayer.Builder().nIn(5*5).nOut(5).build(), "input", "l1")
    .setOutputs("L2")  // We need to specify the network outputs and their order
    .build();

ComputationGraph net = new ComputationGraph(conf);
net.init();
```

Multiple Inputs and Merge Vertex

```java
ComputationGraphConfiguration conf = new NeuralNetConfiguration.Builder()
    .learningRate(0.01)
    .graphBuilder()
    .addInputs("input1", "input2")
    .addLayer("L1", new DenseLayer.Builder().nIn(3).nOut(4).build(), "input1")
    .addLayer("L2", new DenseLayer.Builder().nIn(3).nOut(4).build(), "input2")
    .addVertex("merge", new MergeVertex(), "l1", "l2")
    .addLayer("out", new OutputLayer.Builder().nIn(4+4).nOut(3).build(), "merge")
    .setOutputs("out")
    .build();
```

Multi Task Learning

```java
ComputationGraphConfiguration conf = new NeuralNetConfiguration.Builder()
    .learningRate(0.01)
    .graphBuilder()
    .addInputs("input")
    .addLayer("L1", new DenseLayer.Builder().nIn(3).nOut(4).build(), "input")
    .addLayer("out1", new OutputLayer.Builder()
        .lossFunction(lossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
        .nIn(4).nOut(3).build(), "l1")
    .addLayer("out2", new OutputLayer.Builder()
        .lossFunction(lossFunctions.LossFunction.MSE)
        .nIn(4).nOut(2).build(), "l1")
    .setOutputs("out1", "out2")
    .build();
```

See: http://deeplearning4j.org/compgraph
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)

- Future additions

Section 4 / The End.
(Hadoop) and Spark Integration
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)

- Future additions

Section 4 / The End.
BUT HOW IS THIS CODE DISTRIBUTED?

- So far it is not – only the datasets and interactions are.
- Let's take a look at the source code to proof this.
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

Master receives application

Master waiting for worker update

Receives Update

Ansewers Request

Receives Update

Startup Worker 1

Worker 1 finishes and sends Update

Worker 'waits' (asks) for new update

Worker 1 receives update

Startup Worker 2

Worker 2 finishes and sends Update

Worker 'waits' (asks) for new update

Worker 2 receives update
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

1. Startup Worker 1
   - Master receives application
   - Master waiting for worker update
   - Receives Update
   - Answers Request
   - Receives Update

2. Worker 1 finishes and sends Update
   - Answers Request
   - Receives Update

3. Worker 'waits' (asks) for new update
   - Answers Request
   - Receives Update

4. Worker 1 receives update

5a. Answers Request
5b. Answers Request

6. Worker 2 finishes and sends Update
   - Answers Request
   - Receives Update

   Worker 'waits' (asks) for new update

   Worker 2 receives update
Worker main function after it has been first started:

```java
// Do some work
currentState = WorkerState.STARTED;
//LinkedList<T> records = new LinkedList<T>();

int countTotal = 0;
int countCurrent = 0;
int currentIteration = 0;
int lastUpdate = 0;

computable.setRecordReader(recordParser);

for (currentIteration = 0; currentIteration < workerConf.getIterations(); currentIteration++) {
    //while (doIterations) {
    LOG.debug("Beginning iteration " + (currentIteration +1) + "/" + workerConf.getIterations());

    synchronized (currentState) {
        currentState = WorkerState.RUNNING;
    }

    long mWorkerStart = System.currentTimeMillis();
    T workerUpdate = computable.compute();
    mWorkerExecutions++;
    mWorkerTime += (System.currentTimeMillis() - mWorkerStart);
```

The worker initializes and computes something during the first iteration with the given model on its data, the result is some kind of update.
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

- Master receives application
- Master waiting for worker update
- Startup Worker 1
  - Worker 1 finishes and sends Update
  - Worker 'waits' (asks) for new update
  - Worker 1 receives update
- Startup Worker 2
- Worker 2 finishes and sends Update
- Worker 'waits' (asks) for new update
- Worker 2 receives update
Worker main function after it has been first started:

```java
try {
    synchronized (currentState) {
        ByteBuffer bytes = workerUpdate.toBytes();
        bytes.rewind();

        LOG.info("Sending an update to master");
        currentState = WorkerState.UPDATE;
        if (!masterService.update(workerId, bytes))
            LOG.warn("The master rejected our update");

        mUpdates++;
    }
} catch (AvroRemoteException ex) {
    LOG.error("Unable to send update message to master", ex);
    return -1;
}
```

The worker then sends the update back to the master.
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

- Master receives application
- Master waiting for worker update
- Receives Update
- Ansewrs Request
- Receives Update
- Ansewrs Request
- Worker 1 finishes and sends Update
- Worker 'waits' (asks) for new update
- Worker 1 receives update
- Worker 'waits' (asks) for new update
- Worker 2 receives update
- Worker 2 finishes and sends Update
- Worker 2 receives update
Once the master receives an update from the worker and checks if all of the workers have finished the iterated (that is sent updates). If so, it starts integrating the immediate results and updates the internal update count.

```java
synchronized (workersState) {
    workersUpdate.put(workerId, update);
    workersState.put(workerId, WorkerState.UPDATE);

    // Our latch should have the number of currently active workers
    if (workersUpdate.size() == expectedUpdates.get()) {
        LOG.info("Received updates from all workers, spawning local compute thread");

        // Fire off thread to compute update
        // TODO: need to fix this to something more reusable
        Thread updateThread = new Thread(new Runnable() {

            @Override
            public void run() {
                long startTime, endTime;

                startTime = System.currentTimeMillis();
                T result = computable.compute(workersUpdate.values(),
                                              masterUpdates.values());
                endTime = System.currentTimeMillis();

                LOG.info("Computed local update in " + (endTime - startTime) + "ms");
                expectedUpdates.set(workersCompleted.get().getCount());
            }
        });
    }
}
```

Just fullfilled if all workers are done with the current iteration.
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

1. Master receives application
2. Master waiting for worker update
3. Receives Update
4. Worker 'waits' (asks) for new update
5a. Answers Request
5b. Receives Update
6. Worker 1 receives update
7. Worker 1 finishes and sends Update
8. Worker 2 finishes and sends Update
9. Answers Request
10. Worker 2 receives update
SOURCE CODE (MR/YARN)

Worker after having sent an update (within its main function)

```java
// Wait on master for an update
int nextUpdate;

try {
    LOG.info("Completed a batch, waiting on an update from master");
    nextUpdate = waitOnMasterUpdate(lastUpdate);
} catch (InterruptedException ex) {
    LOG.warn("Interrupted while waiting on master", ex);
    return -1;
} catch (AvroRemoteException ex) {
    LOG.error("Got an error while waiting on updates from master", ex);
    return -1;
}

// Time to get an update
try {
    ByteBuffer b = masterService.fetch(workerId, nextUpdate);
    b.rewind();
    T masterUpdate = updateable.newInstance();
    masterUpdate.fromBytes(b);
    computeable.update(masterUpdate);
}
```

The worker goes over to wait for a next update ...
Worker after having sent an update (waitonmasterupdate)

```java
private int waitOnMasterUpdate(int lastUpdate) throws InterruptedException, AvroRemoteException {
    int nextUpdate = 0;
    long waitStarted = System.currentTimeMillis();
    long waitingFor = 0;

    while ((nextUpdate = masterService.waiting(workerId, lastUpdate, waitingFor)) < 0) {

        synchronized (currentState) {
            currentState = WorkerState.WAITING;
        }

        Thread.sleep(updateSleepTime);
        waitingFor = System.currentTimeMillis() - waitStarted;

        LOG.info("Waiting on update from master with lastID " + lastUpdate + " for " + waitingFor + "msec");
        mWaits++;
    }

    mWaitTime += waitingFor;
    return nextUpdate;
}
```

... and it continues to send periodical wait requests to the master service until it receives a positive number as next update number.
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

1. Master receives application
2. Master waiting for worker update
3. Receives Update
4. Ansewrs Request
5a. Receives Update
5b. Ansewrs Request
6. Worker 'waits' (asks) for new update
7. Worker 1 finishes and sends Update
8. Worker 2 finishes and sends Update
9. Worker 1 receives update
10. Worker 2 receives update
11. Worker 'waits' (asks) for new update
12. Worker 2 receives update
The master service registers the worker as waiting and gives back the current update Id.

```
@Override
public int waiting(WorkerId workerId, int lastUpdate, long waiting) throws AvroRemoteException {
    synchronized (workersState) {
        workersState.put(workerId, WorkerState.WAITING);
        LOG.info("Got waiting message" + ", workerId=" + Utils.getWorkerId(workerId) + ", workerState=" + workersState.get(workerId) + ", lastUpdate=" + lastUpdate + ", currentUpdateId=" + currentUpdateId + ", waitingFor=" + waiting);
    }
    if (MasterState.UPDATING == masterState && lastUpdate == currentUpdateId) return -1;
    return currentUpdateId;
}
```
DISTRIBUTION ON MAPREDUCE (NOW DECRAPITATED)

1. Startup Worker 1
2. Worker 1 finishes and sends Update
3. Receives Update
4. Worker 'waits' (asks) for new update
5. Answers Request
6. Worker 1 receives update
7. Receives Update
8. Answers Request
9. Worker 2 finishes and sends Update
10. Worker 'waits' (asks) for new update
11. Worker 2 receives update
If there is a new update, it fetches some kind of data (most probably related to the model parameters such as weights) from the master server for the current update.
Similar to Mapreduce/Yarn, only that do not have to configure the worker separately.
The master service registers the worker as waiting and gives back the current update Id.

```java
public MultiLayerNetwork fitDataSet(JavaRDD<DataSet> rdd) {
    int iterations = conf.getConf(0).getNumIterations();
    log.info("Running distributed training: (averaging each iteration = " +
             averageEachIteration + " iterations + ") , (num partitions = " +
             rdd.partitions().size() + ")");
    if(!averageEachIteration) {
        //Do multiple iterations and average once at the end
        runIteration(rdd);
    } else {
        //Temporarily set numIterations = 1. Control numIterations externally here so we can average by batch
        for(NeuralNetConfiguration conf : this.conf.getConfs()) {
            conf.setNumIterations(1);
        }
        //Run learning, and average at each iteration
        for(int i = 0; i < iterations; i++) {
            runIteration(rdd);
        }
        //Reset number of iterations in config
        if(iterations > 1) {
            for(NeuralNetConfiguration conf : this.conf.getConfs()) {
                conf.setNumIterations(1);
            }
        }
    }
}
```
Seems like we are spreading some params and an 'updater' to the workers.
RDD's?
We know them already! - Quick lookback:

Left: The initial outlook after reading data from the HDFS. The data from each DataNode will be loaded into local Memory partitions, each Block corresponds to one partition. Middle: After applying some function (one to one mapping), the outcome is not computed but only the nature of the function and the reference to the source RDD is saved (upper row). Right: A function requiring a shuffle is executed, that means the data is differentially distributed among the worker nodes. Locality is lost and if we want to work with HadoopRDD and ShuffledRDD in combination, both need to be kept in Memory. Usually after a shuffle, the current RDD is cached, that is evaluated and stored in memory for future use. The key value pairs depicted here could be any serializable object.
Source Code (Spark Integration)

Run an iteration ... (2)

```java
GradientAdder a = new GradientAdder(paramsLength);
resultsGradient.foreach(a);
INDArray accumulatedGradient = a.getAccumulator().value();
boolean divideGrad = sc.getConf().getBoolean(DIVIDE_ACCUM_GRADIENT, false);
if (divideGrad) {
    maxRep = results.partitions().size();
    accumulatedGradient.divi(maxRep);
}
log.info("Accumulated parameters");
log.info("Summed gradients.");
network.setParameters(network.params(false).addi(accumulatedGradient));
log.info("Set parameters");
JavaDoubleRDD scores = results.mapToDouble(new ScoreMappingG());
lastScore = scores.mean();
if (!initDone) {
    JavaDoubleRDD sm = results.mapToDouble(new SMappingG());
    maxSm = sm.mean().longValue();
}

log.info("Processing updaters");
JavaRDD<Updater> resultsUpdater = results.map(new UpdaterFromGradientTupleFunction());
UpdaterAggregator aggregator = resultsUpdater.aggregate(
    resultsUpdater.first().getAggregator(false),

... and do even more parallel stuff. In the end integrate the results (from all workers) – that is, we also have to wait here until each worker has finished.
```
Iterative Reduce (2)

- Does this even make sense if we need to wait always for the completion of all worker nodes?

---

Stan Kladko, Co-Founder of GalacticExchange, a VC-backed Deep Learning startup
2.6k Views

No.

Spark is significantly inefficient for deep learning and is not so easy to learn.

Implementing Deep Learning algorithms with Spark is awkward. So you get neither simplicity nor performance.

The best way to do Deep Learning is to use a GPU enabled library such as Theano.

Then if you want to build a cluster, use simple old-fashioned tools such as Torque/OpenMPI.

Written 18 Jan • View Upvotes
DL4J COMPONENTS

- Open Source Deep Learning Library
  What is DeepLearning and which part of it is covered by DL4j? (Section 1)

- CPU & GPU support

- Hadoop-Yarn (MR) & Spark integration
  What components are interfaced? (Section 2)
  How are the algorithms distributed? (Section 3)

- Future additions

Section 4 / The End.
OUTLOOK

• Google DistBelief + Apache Horn

Deeplearning4j Roadmap

These priorities have been set by what the Skymind has seen demand for among clients and open-source community members. Contributors are welcome to add features whose priority they deem to be higher.

High priority:

• CUDA rewrite for ND4J (under way)
• CPU optimizations (C++ backend)
• Hyperparameter optimization (underway, basics done: Arbiter)
  • Parameter server
• Sparse support for ND4J
• Performance tests for network training vs. other platforms (and where necessary: optimizations)
• Performance tests for Spark vs. local (ditto)
• Building examples at scale

Medium priority:

• OpenCL for ND4J
• CTC RNN (for speech etc.)
Figure 1: An example of model parallelism in DistBelief. A five layer deep neural network with local connectivity is shown here, partitioned across four machines (blue rectangles). Only those nodes with edges that cross partition boundaries (thick lines) will need to have their state transmitted between machines. Even in cases where a node has multiple edges crossing a partition boundary, its state is only sent to the machine on the other side of that boundary once. Within each partition, computation for individual nodes will be parallelized across all available CPU cores.

Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters \( w \) and push gradients \( \Delta w \) to the parameter server. Right: Sandblaster L-BFGS. A single ‘coordinator’ sends small messages to replicas and the parameter server to orchestrate batch optimization.
2012, DistBelief by Jeff Dean (Google)
2013, Caffe by Yangqing jia (UC Berkeley)
2014, Deeplearning4J by Adam Gibson
2014, DeepDist by Dirk Neumann (Facebook)
...
Apache Horn
a Large-scale Deep Learning
The diagram illustrates a distributed computing system using the BSP (Bulk Synchronous Parallel) framework on Apache Hama or YARN clusters.

- **Client and Web UI**:
- **Apache Horn**:
- **BSP framework on Apache Hama or YARN clusters**:
- **Hadoop HDFS**:

The system involves tasks labeled as Task 1, Task 2, Task 3, Task 4, Task 5, Task 6, Task 7, Task 8, and Task 9. Tasks 2 and 3 are connected to parameter servers, indicating some form of parameter swapping.

The diagram also highlights data parallelism and model parallelism, with arrows indicating the flow of data and tasks.