Deep Learning with Spark

Dr Kashif Rasul\textsuperscript{1} and Shoaib Burq\textsuperscript{2}

Zürich $\rightarrow$ Apache Spark Meetup
25.04.2017

\textsuperscript{1} https://research.zalando.com/, Twitter: @krasul
\textsuperscript{2} http://geografia.com.au, Twitter: @sabman
Agenda

• Introduction to deep learning
• DeepLearning4J
• Distributed training
• Prototyping in Python
• Summary
Deep learning (DL)

- Subfield of machine learning
- Concerned with learning increasingly meaningful representations
- Modern methods involve tens or even hundreds of successive layers of representation
- All learned from exposure to lots of training data
Data driven approach

- Problem: mapping images e.g. 🐱 to label "cat"
- Data driven approach consists of:
  1. **Score** (or prediction): our deep learning model
  2. **Loss**: a measure of our model's performance
  3. **Optimization**: a way to change our model to minimize the loss
Linear case

• Data: \( \{x_i, y_i\}_{i=1}^n \), \( y_i \)s consisting of \( k \) distinct labels

• Score: \( f(x_i; W) = Wx_i \)

• Loss: \( L = \frac{1}{n} \sum_{i=1}^n L_i + \lambda R(W) \)

• Optimization: change \( W \) in the direction of \(-\partial L/\partial W\) to find the optimal \( W \)
Goal: finding the right values for these weights

Input X

Layer (data transformation)

Layer (data transformation)

Predictions $Y'$
DL hype?

- Offers better performance on many problems, especially for computer vision, audio and text tasks
- Automates "feature engineering"
- Advances in:
  1. hardware
  2. datasets and benchmarks
  3. algorithms
DL frameworks

- Collections of many types of layers
- Composition API via a computational graph (values or tensors flow from source to the end)
- Automatic differentiation of each node to implement backpropagation
- APIs to run the optimization on a predefined model or graph with training data and labels
CIFAR-10

- 32x32 pixel RGB images
- 10 classes: 🛫, 🚗, 🦁, 🐶, 🐰, 🐸, 🐪, 🦌, 🐦, 🚗, and 🚗
- 50,000 training images
- 10,000 test images
Convolutional Networks (ConvNets)

• Convolutional layers arranged in 3 dimension: width, height, depth
• The neurons in a layer will only be connected to a small region of the layer before it
• ConvNets transform a 3D volume to another 3D volume
Intuition

• Convolutional layer's weights consist of small learnable filters
• The filter is small spatially, but extends through the full depth of input volume
• We slide across input volume producing a 2-dim activation map of that filter
• As we slide the filter: we are computing the dot product between the filter and the input
• Want to learn filters that activate when they see some specific type of feature at some spatial position in the input
• Stacking these maps for all 5x5x3 filters (6 for this layer) along the depth forms the full output volume
• This process is differentiable (it's also a convolution)
DeepLearning4J (DL4J)

- Java based DL framework
- Multi-GPU (NVIDIA) support
- Using spark: to parallelize via "data parallelism"
- Import Keras models
- Helper libraries and sample code on github
cifarTrain = new CifarDataSetIterator(batchSize,...);
cifarTest = ...

MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    ... // score, loss and optimization configuration here

MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();

for( int i=0; i<nEpochs; i++ ) {
    model.fit(cifarTrain);
    // evaluate performance on cifarTest
    ...
}
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
  .seed(seed)
  .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
  .iterations(1)
  .activation(Activation.LEAKYRELU)
  .weightInit(WeightInit.XAVIER)
  .learningRate(0.02)
  .updater(Updater.NESTEROVS).momentum(0.9)
  .regularization(true).l2(1e-4)
  .list()
  .layer(0, new DenseLayer.Builder().nIn(32 * 32 * 3).nOut(500).build())
  .layer(1, new DenseLayer.Builder().nIn(500).nOut(100).build())
  .layer(2, new OutputLayer.Builder(
      LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
    .activation(Activation.SOFTMAX).nIn(100).nOut(10).build())
  .pretrain(false).backprop(true)
  .build();
layer(1, new ConvolutionLayer.Builder(3, 3)
    .nIn(channels)
    .padding(1, 1)
    .nOut(64)
    .weightInit(WeightInit.RELU)
    .activation(Activation.LEAKYRELU)
    .build())
.layer(2, new SubsamplingLayer.Builder(
    SubsamplingLayer.PoolingType.MAX)
    .kernelSize(2, 2)
    .build())
.layer(3, new ConvolutionLayer.Builder(3, 3)...)
...
Stochastic gradient descent (SGD)

- Vanilla optimization: update the weights with respect to all the data
- Vanilla SGD: iteratively update the weights with respect to a small random batch of data (batchSize)
- After an update has seen all the data we mark it as an epoch
- Fancier SGD methods use momentum terms etc.
- Sequential process
SparkConf sparkConf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(sparkConf);

cifarTrain = new CifarDataSetIterator(batchSizePerWorker,...);

List<DataSet> trainDataList = new ArrayList<>();
while (cifarTrain.hasNext()) {
    trainDataList.add(cifarTrain.next());
}

JavaRDD<DataSet> trainData = sc.parallelize(trainDataList);
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    ...

TrainingMaster tm = new ParameterAveragingTrainingMaster.Builder(batchSizePerWorker)
    .averagingFrequency(5)
    .workerPrefetchNumBatches(2)
    .batchSizePerWorker(batchSizePerWorker)
    .build();

SparkDl4jMultiLayer sparkNet = new SparkDl4jMultiLayer(sc, conf, tm);

for (int i=0; i<nEpochs; i++) {
    sparkNet.fit(trainData);
}
CudaEnvironment.getInstance().getConfiguration()
    .allowMultiGPU(true)
    .setMaximumDeviceCache(2L * 1024L * 1024L * 1024L)
    .allowCrossDeviceAccess(true);

MultiLayerConfiguration conf = new NeuralNetConfiguration...
MultiLayerNetwork model = new MultiLayerNetwork(conf);

ParallelWrapper wrapper = new ParallelWrapper.Builder(model)
    .prefetchBuffer(24).workers(4)
    .averagingFrequency(3).useLegacyAveraging(true)
    .build();

for( int i=0; i<nEpochs; i++ ) {
    wrapper.fit(cifarTrain);
}
Keras

- High level API based on python
- Backend: TensorFlow or Theano
- Allows for easy and fast prototyping
- Models are described in Python code
inputs = Input(shape=(784,))

x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
scores = Dense(10, activation='softmax')(x)

model = Model(inputs=inputs, outputs=scores)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels)
# creates a HDF5 file 'my_model.h5'
model.save('my_model.h5')
model = load_model('my_model.h5')

# model reconstruction from JSON:
json_string = model.to_json()
model = model_from_json(json_string)

# save model weights
model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5')
// configuration only
CopyMultiLayerNetworkConfiguration modelConfig =
    KerasModelImport.importKerasSequentialConfiguration
    ("PATH TO YOUR JSON FILE", enforceTrainingConfig);

// configuration and weights
MultiLayerNetwork network =
    KerasModelImport.importKerasSequentialModelAndWeights
    ("PATH TO YOUR HDF5 FILE", enforceTrainingConfig);
Summary

• DL: learning successive "layers" of representations
• Data driven approach: three parts
• Frameworks: collection of layers and a computational graph
• ConvNets: transform 3D volumes to 3D volumes
• DL4J: implements both types of parallelism (data and model)
• Suggestion: prototype in Keras and train in DL4J
Thank you!

Questions?

Checkout http://deeplearningbox.com/