BigDL: Apache Spark* 上の最適化されたディープラーニング

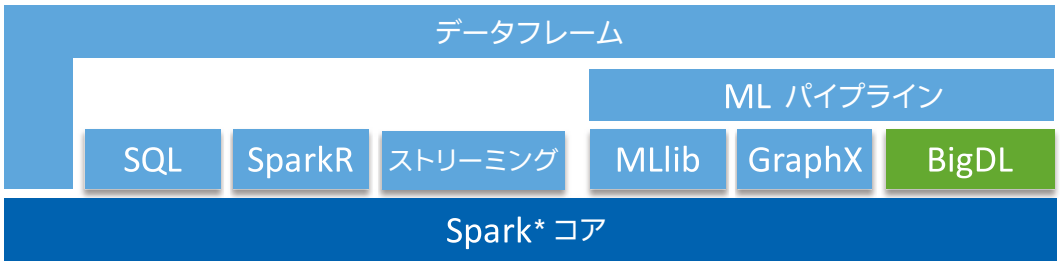
オープンソースの分散型ディープラーニング・フレームワークでディープラーニングの普及を促進

Jason Dai インテル コーポレーション 主任エンジニア
Radhika Rangarajan インテル コーポレーション テクニカル・プログラム・マネージャー

人工知能 (AI) は現代のスマートで接続された世界で中心的な役割を果たしており、ディープラーニング機能を備えた、スケーラブルな分散型ビッグデータ解析の必要性は高まるばかりです。また、特徴量の設計や従来のマシンラーニングをサポートする既存のデータ処理パイプラインとディープラーニングを同じクラスターで処理したいという要望も増えています。ビッグデータの解析とディープラーニング向けの統一プラットフォームを求める声に応えるため、インテルは、Apache Spark* 向けのオープンソース分散型ディープラーニング・フレームワーク、**BigDL** (英語) を最近リリースしました。この記事では、BigDL の機能と BigDL を使用してモデルを作成する方法を紹介します。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

BigDL は Spark* の上位のライブラリーとして実装され (**図 1**)、スケールアウト・コンピューティングを容易に行うことができます。BigDL を利用すると、ディープラーニング・アプリケーションを既存の Spark* または Hadoop* クラスター上で直接実行できる標準 Spark* プログラムとして記述することができます。



1 BigDL の実装

BigDL の概要

BigDL は、次の特長を備えており、ビッグデータおよび Spark* プラットフォームでディープラーニング機能をネイティブにサポートします。

- **ディープラーニングの包括的なサポート。** Torch を手本に開発された BigDL は、数値計算 (テンソルなど) や高レベルのニューラル・ネットワークを含む、ディープラーニングの包括的なサポートを提供します。さらに、BigDL を使用して、訓練された Caffe* または Torch のモデルを Spark* プログラムにロードすることもできます。
- **非常に高いパフォーマンス。** 高いパフォーマンスが得られるように、BigDL は各 Spark* タスクで**インテル® マス・カーネル・ライブラリー (インテル® MKL)** およびマルチスレッド・プログラミングを使用します。そのため、シングルノードのインテル® Xeon® プロセッサー (メインストリーム GPU と同等) でオープンソースの Caffe*、Torch、TensorFlow* をそのまま利用するよりもはるかに高速です。
- **効率的なスケールアウト。** BigDL は、Apache Spark* を活用して、Spark* での同期 SGD や Allreduce 通信の効率的な実装により、効率的にスケールアウトしてビッグデータ・スケールでデータ解析を実行できます。

Spark* とのネイティブ統合は、BigDL の重要な利点です。BigDL は Spark* の上に構築されるため、ディープラーニングにおいて計算負荷の高いモデルの訓練を簡単に分散できます。計算を明示的に分散するように要求するのではなく、BigDL は Spark* クラスター全体にワークを自動的に分配します。

BigDL は (データストレージ、データ処理 / データマイニング、特徴量の設計、典型的なマシンラーニング、ディープラーニング向けの) 統一データ解析プラットフォームとして Hadoop* と Spark* をサポートし、ビッグデータのユーザーやデータ・サイエンティストがディープラーニングをより簡単に利用できるようにします。

典型的な BigDL のユースケースは次のとおりです。

- **大量のデータ**をその格納場所 (HDFS*、HBase*、Hive* など) と同じビッグデータ (Hadoop* および Spark*) クラスター上でディープラーニング・テクノロジーを使用して解析し、別のシステムとの間の大量の不要なデータ転送を排除します。
- **ディープラーニング機能** (訓練、微調整、予測など) をビッグデータ (Spark*) プログラムやワークフローに追加して、システムの複雑さやエンドツーエンドのレイテンシーを減らします。
- **既存の Hadoop* および Spark* クラスターを活用**してディープラーニング・アプリケーションを実行し、ほかのワークロード (ETL、データ・ウェアハウス、特徴量の設計、典型的なマシンラーニング、グラフ解析など) を動的に共有します。

BigDL を利用する

BigDL は、数値計算 (テンソルなど)、高レベルのニューラル・ネットワーク、分散型確率的最適化 (Spark* での同期ミニバッチ SGD や Allreduce 通信など) を含む、ディープラーニングを包括的にサポートします。**表 1** は、BigDL で提供される抽象化と API をまとめたものです。

名前	説明
Tensor	数値型の多次元配列 (Int、Float、Double など)
Module	ニューラル・ネットワークの層 (ReLU、Linear、SpatialConvolution、Sequential など)
Criterion	指定された入力およびターゲット、指定された損失関数ごとの勾配の計算
Sample	特徴およびラベル (テンソル) からなるレコード
DataSet	データの訓練、検証、テスト、DataSet の (-> 演算子による) 一連のデータ変換には Transformer を使用
Engine	訓練のランタイム環境 (ノード数、コア数、Spark*/ ローカル、マルチスレッド)
Optimizer	ローカルまたは分散型訓練の確率的最適化 (SGD、AdaGrad のようなさまざまな OptimMethod を使用)

表 1. BigDL の抽象化と API

BigDL プログラムは、ローカル Scala*/Java* プログラムとして、または Spark* プログラムとして実行できます。[編集者注 : Python* のサポートはまもなく行われます。この記事が掲載される頃にはサポートされているでしょう。] 対話型の Scala* シェル (REPL) を使用して BigDL コードをローカル Scala*/Java* プログラムとして作成するには、最初に次のように入力します。

```
$ source PATH_To_BigDL/scripts/bigdl.sh
$ SPARK_HOME/bin/spark-shell --jars bigdl-0.1.0-SNAPSHOT-jar-with-dependencies.
jar
```

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

Scala* シェルは次のように表示されます。

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_79)
Spark context available as sc.
scala>
```

BigDL の Tensor API の使用例を次に示します。

```
scala> import com.intel.analytics.bigdl.tensor.Tensor
import com.intel.analytics.bigdl.tensor.Tensor

scala> Tensor[Double](2,2).fill(1.0)
res9: com.intel.analytics.bigdl.tensor.Tensor[Double] =
1.0   1.0
1.0   1.0
[com.intel.analytics.bigdl.tensor.DenseTensor of size 2x2]
```

BigDL の Module API の使用例を次に示します。

```
scala> import com.intel.analytics.bigdl.numeric.NumericFloat // import global
float tensor numeric type
import com.intel.analytics.bigdl.numeric.NumericFloat

scala> import com.intel.analytics.bigdl.nn._
import com.intel.analytics.bigdl.nn._

scala> val f = Linear(3,4) // create the module
mlp: com.intel.analytics.bigdl.nn.Linear[Float] = nn.Linear(3 -> 4)

// let's see what f's parameters were initialized to.('nn' always inits to
something reasonable)
scala> f.weight
res5: com.intel.analytics.bigdl.tensor.Tensor[Float] =
-0.008662592   0.543819     -0.028795477
-0.30469555    -0.3909278    -0.10871882
0.114964925    0.1411745     0.35646403
-0.16590376    -0.19962183    -0.18782845
[com.intel.analytics.bigdl.tensor.DenseTensor of size 4x3]
```

BigDL で単純なテキスト分類器を作成する

では次に、単純な**畳み込みニューラル・ネットワーク (CNN) モデル** (英語) を使用してテキスト分類器を作成する方法を見てみましょう。

BigDL プログラムは com.intel.analytics.bigdl._ をインポートして開始した後、エンジンを初期化します (executor ノードの数、各 executor の物理コアの数、Spark* で実行するかローカル Java* プログラムとして実行するかを含む)。

```
val sc = new SparkContext(
 Engine.init(param.nodeNum, param.coreNum, true).get
  .setAppName("Text classification")
  .set("spark.akka.frameSize", 64.toString)
  .set("spark.task.maxFailures", "1"))
```

次に、事前に訓練された Word Embedding をブロードキャストし、RDD 変換 (vectorizedRdd) を使用して入力データをロードします。

```
 // For large dataset, you might want to get such RDD[(String, Float)] from
HDFS
 val dataRdd = sc.parallelize(loadRawData(), param.partitionNum)
 val (word2Meta, word2Vec) = analyzeTexts(dataRdd)
 val word2MetaBC = sc.broadcast(word2Meta)
 val word2VecBC = sc.broadcast(word2Vec)
 val vectorizedRdd = dataRdd
   .map {case (text, label) => (toTokens(text, word2MetaBC.value), label)}
   .map {case (tokens, label) => (shaping(tokens, sequenceLen), label)}
   .map {case (tokens, label) => (vectorization(
    tokens, embeddingDim, word2VecBC.value), label)}
```

次に、処理したデータ (vectorizedRdd) をサンプル RDD に変換し、サンプル RDD (sampleRDD) を訓練データ (trainingRDD) および検証データ (valRDD) にランダムに分割します。

```
val sampleRDD = vectorizedRdd.map {case (input: Array[Array[Float]],
                     label: Float) =>
  Sample(
   featureTensor = Tensor(input.flatten, Array(sequenceLen, embeddingDim))
    .transpose(1, 2).contiguous(),
   labelTensor = Tensor(Array(label), Array(1)))
 }

val Array(trainingRDD, valRDD) = sampleRDD.randomSplit(
 Array(trainingSplit, 1 - trainingSplit))
```

次に、buildModel を呼び出して CNN モデルを作成します。

```
def buildModel(classNum: Int): Sequential[Float] = {
 val model = Sequential[Float]()
 model.add(Reshape(Array(param.embeddingDim, 1, param.maxSequenceLength)))
 model.add(SpatialConvolution(param.embeddingDim, 128, 5, 1))
 model.add(ReLU())
 model.add(SpatialMaxPooling(5, 1, 5, 1))
 model.add(SpatialConvolution(128, 128, 5, 1))
 model.add(ReLU())
 model.add(SpatialMaxPooling(5, 1, 5, 1))
 model.add(SpatialConvolution(128, 128, 5, 1))
 model.add(ReLU())
 model.add(SpatialMaxPooling(35, 1, 35, 1))
 model.add(Reshape(Array(128)))
 model.add(Linear(128, 100))
 model.add(Linear(100, classNum))
 model.add(LogSoftMax())
 model
}
```

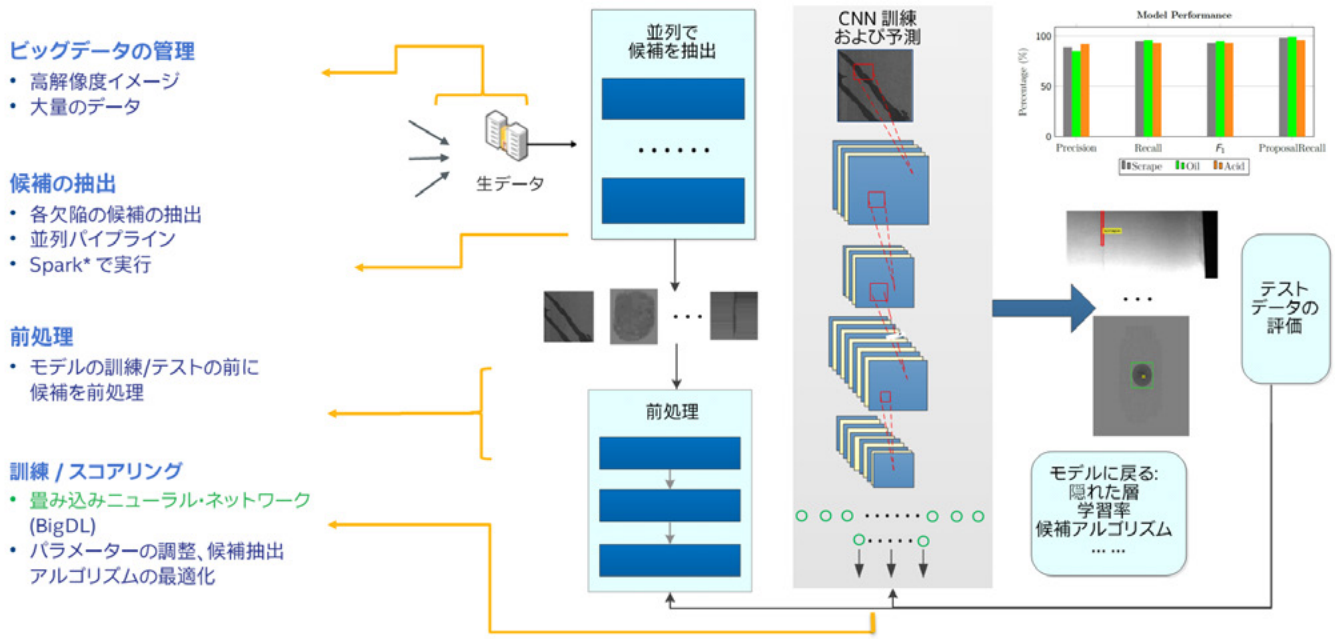
次に、Optimizer を作成し、訓練データ RDD (trainingRDD) を Optimizer に (特定のバッチサイズで) 渡して、(最適化メソッドとして Adagrad を使用し、ステートに関連するハイパーパラメーターを設定して) モデルを最終的に訓練します。

```
val optimizer = Optimizer(
 model = buildModel(classNum),
 sampleRDD = trainingRDD,
 criterion = new ClassNLLCriterion[Float](),
 batchSize = param.batchSize
)
val state = T("learningRate" -> 0.01, "learningRateDecay" -> 0.0002)
optimizer
  .setState(state)
  .setOptimMethod(new Adagrad())
.setValidation(Trigger.everyEpoch, valRDD, Array(new Top1Accuracy[Float]),
        param.batchSize)
  .setEndWhen(Trigger.maxEpoch(2))
  .optimize()
```

BigDL でエンドツーエンド・アプリケーションを作成する

BigDL では、データの管理、特徴の管理、特徴の変換、モデルの訓練および予測、結果の評価を含む、Spark* ベースの単一解析パイプラインを使用して、エンドツーエンドの AI アプリケーションを作成することができます。我々は、異なる複数の分野のユーザーと作業を行い、不正検出や欠陥検出などに BigDL を使用し

たエンドツーエンドのソリューションを開発しました。**図 2** は、Spark* で BigDL を使用して構築された、エンドツーエンドのイメージ認識とオブジェクト検出パイプラインを示しています。このパイプラインは、製造パイプラインから大量のイメージを収集して処理し、(BigDL で畳み込みニューラル・ネットワーク・モデルを使用して) これらのイメージから製品の欠陥を自動的に検出します。



2 エンドツーエンドのイメージ認識とオブジェクト検出パイプライン

ディープラーニングの普及を促進

この記事では、Apache Spark* 向けのオープンソース分散型ディープラーニング・フレームワーク、BigDL について説明しました。BigDL は、ディープラーニング・アプリケーションを標準 Spark* プログラムとして記述し、これらのディープラーニング・アプリケーションを既存の Spark* または Hadoop* クラスター上で直接実行できるようにすることで、ビッグデータのユーザーやデータ・サイエンティストがディープラーニングをより簡単に利用できるようにします。その結果、Hadoop*/Spark* を、優れたスケールメリット、高いリソース使用率、使いやすさ / 開発しやすさ、優れた TCO を提供する、データストレージ、データ処理 / データマイニング、特徴量の設計、従来のマシンラーニング、ディープラーニング・ワークロード向けの統一プラットフォームにしているのです。

GitHub* から BigDL を入手する (英語) ›

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。