Atitit 机器学习deeplearning4j 代码范例 艾提拉总结测试

目录

[1.1. Maven 一定要添加 nd4j-nativ 1](#_Toc2877)

[1.2. 代码 MLPMnistSingleLayerExample 2](#_Toc11215)

[1.3. 输出 4](#_Toc31144)

[1.4. 总结 5](#_Toc20765)

[2. 注意事项 5](#_Toc22704)

## Maven 一定要添加 nd4j-nativ

<dependency>

<groupId>org.deeplearning4j</groupId>

<artifactId>deeplearning4j-core</artifactId>

<version>1.0.0-beta2</version>

</dependency>

<dependency>

<groupId>org.nd4j</groupId>

<artifactId>nd4j-api</artifactId>

<version>1.0.0-beta2</version>

</dependency>

<!-- Copy this into your own projects (without the comment block around

it), for CPU operations -->

<!-- -->

<dependency>

<groupId>org.nd4j</groupId>

<artifactId>nd4j-native-platform</artifactId>

<version>1.0.0-beta2</version>

</dependency>

<!-- log -->

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.25</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.25</version>

<!-- <scope>test</scope> -->

</dependency>

## 代码 MLPMnistSingleLayerExample

**package** ml;

**import** org.nd4j.linalg.dataset.api.iterator.DataSetIterator;

**import** org.nd4j.linalg.learning.config.Sgd;

**import** org.deeplearning4j.datasets.iterator.impl.MnistDataSetIterator;

**import** org.deeplearning4j.eval.Evaluation;

**import** org.deeplearning4j.nn.api.OptimizationAlgorithm;

**import** org.deeplearning4j.nn.conf.MultiLayerConfiguration;

**import** org.deeplearning4j.nn.conf.NeuralNetConfiguration;

**import** org.deeplearning4j.nn.conf.Updater;

**import** org.deeplearning4j.nn.conf.layers.DenseLayer;

**import** org.deeplearning4j.nn.conf.layers.OutputLayer;

**import** org.deeplearning4j.nn.multilayer.MultiLayerNetwork;

**import** org.deeplearning4j.nn.weights.WeightInit;

**import** org.deeplearning4j.optimize.listeners.ScoreIterationListener;

**import** org.nd4j.linalg.activations.Activation;

**import** org.nd4j.linalg.api.ndarray.INDArray;

**import** org.nd4j.linalg.dataset.DataSet;

**import** org.nd4j.linalg.lossfunctions.LossFunctions.LossFunction;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

/\*\*

\* A Simple MLP applied to digit classification for MNIST.

\*/

**public** **class** MLPMnistSingleLayerExample {

**private** **static** Logger *log* = LoggerFactory.*getLogger*(MLPMnistSingleLayerExample.**class**);

**public** **static** **void** main(String[] args) **throws** Exception {

*log*.info("Train model start....");

// org.nd4j.nativeblas.LongPointerWrapper

**final** **int** numRows = 28;// 图像宽

**final** **int** numColumns = 28;// 图像长

**int** outputNum = 10;// 输出的类别数

**int** batchSize = 128;// 没128个样本参加训练

**int** rngSeed = 123;//

**int** numEpochs = 1;// 训练集样本迭代的次数

// load dataq

// Get the DataSetIterators:

DataSetIterator mnistTrain = **new** MnistDataSetIterator(batchSize, **true**, rngSeed);

DataSetIterator mnistTest = **new** MnistDataSetIterator(batchSize, **false**, rngSeed);

*log*.info("Build model....");

MultiLayerConfiguration conf = **new** NeuralNetConfiguration.Builder().weightInit(WeightInit.***XAVIER***)

.activation(Activation.***RELU***)

.optimizationAlgo(OptimizationAlgorithm.***STOCHASTIC\_GRADIENT\_DESCENT***).updater(**new** Sgd(0.05))

// .iterations(1)

// .learningRate(0.006)//学习率

// // ... other hyperparameters

.list().layer(0, **new** DenseLayer.Builder()// 第一层网络配置

.nIn(numRows \* numColumns)// 输入数目

.nOut(1000)// 输出数目

.activation(Activation.***RELU***)//激活函数 relu

.weightInit(WeightInit.***XAVIER***)// 权值初始化

.build())

// 输出层指定误差函数

.layer(1, **new** OutputLayer.Builder(LossFunction.***NEGATIVELOGLIKELIHOOD***)// 误差函数

.nIn(1000)// 输入

.nOut(outputNum)// 输出

.activation(Activation.***SOFTMAX***)//激活函数

.weightInit(WeightInit.***XAVIER***).build())

.pretrain(**false**).backprop(**true**).build();

MultiLayerNetwork model\_MultiLayerNetwork = **new** MultiLayerNetwork(conf);

model\_MultiLayerNetwork.init();

model\_MultiLayerNetwork.setListeners(**new** ScoreIterationListener(1));

*log*.info("Train model....");

**for** (**int** i = 0; i < numEpochs; ++i) {

model\_MultiLayerNetwork.fit(mnistTrain);

*log*.info("\*\*\* Completed epoch " + i + "\*\*\*");

*log*.info("Evaluate model....");

Evaluation eval = **new** Evaluation(outputNum);

**while** (mnistTest.hasNext()) {

DataSet ds = mnistTest.next();

INDArray output = model\_MultiLayerNetwork.output(ds.getFeatures(), **false**);

eval.eval(ds.getLabels(), output);

}

*log*.info(eval.stats());

mnistTest.reset();

}

}

}

## 输出

========================Evaluation Metrics========================

# of classes: 10

Accuracy: 0.9135

Precision: 0.9129

Recall: 0.9125

F1 Score: 0.9123

Precision, recall & F1: macro-averaged (equally weighted avg. of 10 classes)

=========================Confusion Matrix=========================

0 1 2 3 4 5 6 7 8 9

---------------------------------------------------

949 0 1 2 0 7 16 1 4 0 | 0 = 0

0 1105 2 4 1 2 4 1 16 0 | 1 = 1

10 5 902 18 13 0 22 16 40 6 | 2 = 2

3 1 18 921 1 22 4 12 19 9 | 3 = 3

1 2 4 1 917 1 17 1 7 31 | 4 = 4

7 3 2 44 13 761 21 8 25 8 | 5 = 5

7 3 4 2 13 10 915 1 3 0 | 6 = 6

1 17 26 4 13 0 0 925 4 38 | 7 = 7

3 7 8 28 11 22 15 12 858 10 | 8 = 8

8 9 6 11 52 14 1 19 7 882 | 9 = 9

Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times

## 总结

只迭代一次就基本90的准确率了。。太多了碟台太花时间，一次迭代就8s了。。。

# 注意事项

nd4j-nativ 版本一定和主板吧一致。下载maven狠话时间的