# Deeeplearning4j 指南

English version: https://www.baeldung.com/deeplearning4j

## 1. 介绍

DeepLearning4j(dl4j)是使用 Java 进行机器学习的十分流行且功能强大的库。我们将在本文中使用 DeepLearning4j(dl4j)的 java 库来建立一个简单的神经网络。

开始之前需要提醒大家的是：本文不需要你深入了解“线性代数”、“统计学”、“机器学习理论”和机器学习工程师掌握的其它很多知识，你一样能轻松学习和理解本文中的知识。

## 2. 什么是深度学习？

神经网络的深度学习是一种计算模型，该模型中很多层(layer)相互连接，并且每一层都有很多节点（node）。

每个节点（node）都是一个类似神经元的数字处理器。一个节点首先获取输入数据，将这些数据进行处理（包括设置权重和函数计算），然后将处理结果传送至输出节点。这样的神经网络可以用人们提供的数据源样例来进行训练。

训练的主要目的是让节点保存一些数字（权重），这些数字用于以后调整学习到的计算公式。训练样本会包括有某些特征的数据项和这些数据项的预知分类（比如：一个 16*16 像素的图是一个手写的字母“a”）。

训练结束以后，神经网络就可以用全新的数据来生成结果。就算神经网络从来都没有看见过这些全新数据，也同样可以生成结果。一个好的模型加上有效训练过而形成的神经网络，可以识别图像、手写文字、语音、处理统计数据并将结果用于商业智能，还有其它更多的场景。

深度神经网络在最近几年变得流行，是因为高性能计算和并行计算得到长足进步。深度神经网络不同于简单神经网络，它由很多中间层（隐藏层）组成。这种结构能够让神经网络用更复杂的方式（比如：递归、循环、卷积等等）处理数据。从数据中抽取出更多的信息。

## 3. 创建工程

为了使用 deeplearning4j 库，我们用到的 java 最低版本是 Java 7。并且由于 dl4j 会引用其它机器学习的本地实现，所以只能使用 64 位的 JVM 版本。

我们可以通过下面的命令来检查 java 是否满足需求：

```
$ java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java Hotspot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

首先，往 pom.xml 文件中添加需要的库文件。我们把库文件的版本号提取到属性项里面去（最新的版本号，查看 Maven Central 仓库）

```
<properties>
    <dl4j.version>0.9.1</dl4j.version>
</properties>
```

```
<dependencies>
    <dependency>
        <groupId>org.nd4j</groupId>
        <artifactId>nd4j-native-platform</artifactId>
        <version>${dl4j.version}</version>
    </dependency>

    <dependency>
        <groupId>org.deeplearning4j</groupId>
        <artifactId>deeplearning4j-core</artifactId>
        <version>${dl4j.version}</version>
    </dependency>
</dependencies>
```

请注意 *nd4j-native-platform* 是神经网络深度学习的多个实现中的其中一个。
它依赖于不同平台（比如：macOS，windows，Linux，Android 等等）的本地库。如果我们想使用显卡来参与计算，可以切换成 *nd4j-cuda-8.0-platform* 版本，它支持 CUDA 计算模型。

## 4. 准备数据

### 4.1 准备数据文件

我们将实现一个机器学习版本的 Hello World（鸢尾花的分类）。这个数据集采集了不同品种的花（鸢尾，杂色鸢尾和弗吉尼亚鸢尾）。

这些不同品种的鸢尾花，它们的花瓣和萼片的长度和宽度不一样。如果要写出一个精确的算法来分类这些数据非常困难（比如：判断哪一种花属于什么品种）。但可以容易的实现一个训练良好的神经网络，进行快速、且极少差错的将它们分类出来。

我们将使用 CSV 格式的数据，它的 0 至 3 列包含了花品种的不同特征，第 4 列包含了这条数据属于哪个品种，品种以编码的形式记录为 0,1,2：

```
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
…
7.0,3.2,4.7,1.4,1
6.4,3.2,4.5,1.5,1
6.9,3.1,4.9,1.5,1
…
```

### 4.2 向量化读取数据

我们把类别转换为数字是因为神经网络只能处理数字。将真实世界中的数据转换成一串数字（向量）称为向量化 -- deeplearning4j 使用 datavec 库来处理。

首先，我们使用该库将文本文件转换成向量数据。新建 *CSVRecordReader* 对象的时候，我们需要指定需要跳过的行数（比如：该文件包括标题行）和分隔符号（这里我们使用的是逗号）：

```
try (RecordReader recordReader = new CSVRecordReader(0, ',')) {
  recordReader.initialize(new FileSplit(
    new ClassPathResource("iris.txt").getFile()));
```

```
4
5       // …
6   }
```

我们可以使用 *DataSetIterator* 接口的多个方法实现来遍历数据集。如果数据集非常巨大，使用它提供的分页功能和缓存功能就非常方便。

因为我们的数据集很小只有 150 条数据，所以我们调用 *iterator.next()*方法将它们一次性的读入内存。

我们也指定了鸢尾花品种所在的列，代码里是 FEATURES_COUNT（4）和类别的总数 CLASSES_COUNT（3）。

同时，需要注意我们必须打乱数据，从而避免原始文件里面使用品种排过序。

我们指定随机数种子为 42，而不是默认的 *System.currentTimeMillis()*方法产生的不同的数，这样我们每次打乱数据后结果都会一样，使得我们每次运行程序得到的结果都一样。

```
1   DataSetIterator iterator = new RecordReaderDataSetIterator(
2    recordReader, 150, FEATURES_COUNT, CLASSES_COUNT);
3   DataSet allData = iterator.next();
4   allData.shuffle(42);
```

### 4.3 格式化和拆分

在进行训练之前，我们还需要对数据进行格式化。格式化分为两个阶段：

. 获取数据的一些统计数据（fit）

. 将数据通过一些方式变化（转换）成统一风格

格式化对不同类型的数据是不同的。

比如，如果我们要处理不同尺寸的图像，我们需要首先收集图像尺寸的统计数据，然后将他们缩放成统一的尺寸。

但是对数字而言，格式化一般都是指转换成符合正态分布。*NormalizerStandardize* 类可以帮到我们：

```
1   DataNormalization normalizer = new NormalizerStandardize();
2   normalizer.fit(allData);
3   normalizer.transform(allData);
```

现在数据已经准备就绪，我们需要把数据集拆分成两个部分。

第一部分用于以后的训练部分。我们使用第二部分（神经网络完全没有见到过这部分数据）来测试训练好的神经网络。

它就可以让我们验证分类的正确性。我们取其中 65%的数据(0.65)用来训练，留下剩下的 35%的数据用于测试:

```
1   SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.65);
2   DataSet trainingData = testAndTrain.getTrain();
3   DataSet testData = testAndTrain.getTest();
```

## 5．准备神经网络配置

### 5.1 流式配置构建器

现在可以使用流式构建器来构建我们的神经网络：

```
1    MultiLayerConfiguration configuration
2      = new NeuralNetConfiguration.Builder()
3        .iterations(1000)
```

```
4         .activation(Activation.TANH)
5         .weightInit(WeightInit.XAVIER)
6         .learningRate(0.1)
7         .regularization(true).l2(0.0001)
8         .list()
9         .layer(0, new DenseLayer.Builder().nIn(FEATURES_COUNT).nOut(3).build())
10        .layer(1, new DenseLayer.Builder().nIn(3).nOut(3).build())
11        .layer(2, new OutputLayer.Builder(
12         LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
13           .activation(Activation.SOFTMAX)
14           .nIn(3).nOut(CLASSES_COUNT).build())
15        .backprop(true).pretrain(false)
16        .build();
```

虽然我们使用了简化版本的流式构建器来构造神经网络模型，也有很多知识需要消化，也有很多参数可以调整。下面我们把该模型分开来说。

### 5.2 设置神经网络参数

这里的 iterations()构造方法指定了迭代优化的次数。

这里迭代优化的意思是对训练集执行多次遍历，直到神经网络收敛到良好的结果为止。

通常，当我们训练真实的大数据集时，我们使用多个 epochs(所有的数据全部通过了网络)，每一个 epoch 作为一个迭代。但是因为我们的初始数据集很小，我们使用一个 epoch 和多个迭代。

这里的 activation() 是一个运行在节点内部的函数，该函数决定了输出数据。

最简单的激活函数可以是线性的 f(x)=x。但是事实表明只有非线性函数才能让神经网络使用多个节点从而解决复杂任务。

deeplearning4j 预设了很多不同的激活函数，可以在 *org.nd4j.linalg.activations.Activation* 枚举类中找到。如果必要的话，我们也可以编写自己的激活函数。但是我们这里直接使用预设的 tanh 双曲函数。

这里的 *weightInit()*也可以从多种方式中选择一个，用来设置神经网络的初始化权重。正确的初始化权重可以显著的影响训练结果。为了不深究从数学知识，我们直接设置成高斯分布(WeightInit.XAVIER)的形式，这个设置是学习阶段很好的选择。

Learning rate(学习速度) 是一个关键参数，它将显著的影响神经网络的学习能力。

在复杂的情况下，我们需要花很长时间来调整该参数。但是对于我们现在这个简单任务，我们使用非常明显的值 0.1，并且通过 *learningRate()*方法来设置。

训练神经网络中很多问题中的一个是过度拟合，如果神经网络“记住”了训练数据的时候会发生过度拟合。

当神经网络为训练数据设置过分高的权重，这是对其它新数据反而会产生很差的结果，这时就发生了过度拟合。

为了解决这个问题，我们设置了 L2 正则化，代码为*.regularization(true).l2(0.0001)*。这样当正则化在太大权重时会“惩罚”神经网络，从而避免过度拟合。

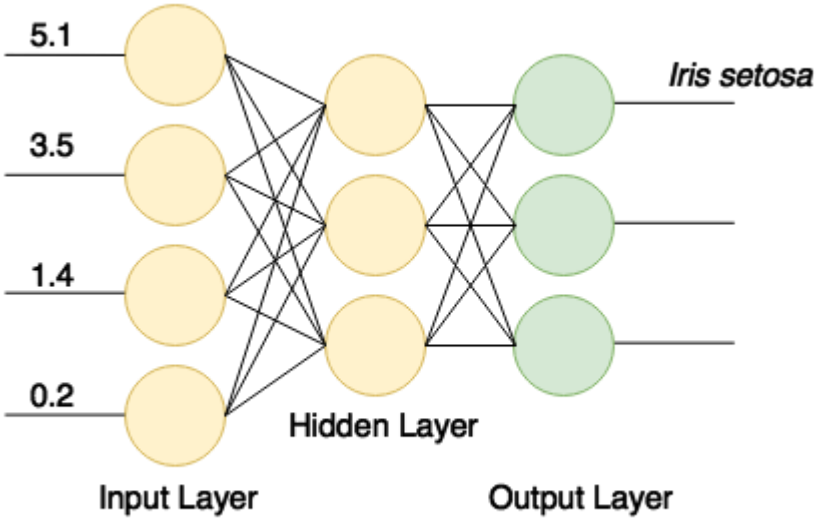
### 5.3 创建神经网络层级

下一步，我们开始创建神经网络的密集（也称为全连接）层级。

第一层需要包含跟训练数据中的列中相同多数量(4)的节点。

第二个密集层包含三个节点。这个数值是可以调整的。但是输出层和它的前一层的数量必须一样。

最后是输出层需要包含的节点数量必须跟鸢尾花品种的数量（3）一样。神经网络结果如下图所示：




当成功完成训练以后，我们将拥有一个神经网络，它输入参数是 4 个值，然后发送一个信号到三个输出节点中的一个。这就是简单分类神经网络。

创建神经网络的最后一步，我们通过*.backprop(true).pretrain(false)*.设置了反向传播（一个最有效的训练方法）并且禁止了预训练。

**6．创建和训练网络**

现在我们开始通过配置好的 configuration 来创建神经网络，初始化它并运行它：

```
1    MultiLayerNetwork model = new MultiLayerNetwork(configuration);
2    model.init();
3    model.fit(trainingData);
```

下面我们可以使用剩下的数据集测试这个训练模型，并且通过评估 3 个品种的指标来验证结果：

```
1    INDArray output = model.output(testData.getFeatureMatrix());
2    Evaluation eval = new Evaluation(3);
3    eval.eval(testData.getLabels(), output);
```

我们通过打印 ***eval.stats()***，可以看到我们的神经网络对鸢尾花的分类工作表现很好，仅仅对分类 1 和分类 2 错误识别了 3 次。

```
1     Examples labeled as 0 classified by model as 0: 19 times
2     Examples labeled as 1 classified by model as 1: 16 times
3     Examples labeled as 1 classified by model as 2: 3 times
4     Examples labeled as 2 classified by model as 2: 15 times
5
6     ==========================Scores========================================
7     # of classes: 3
```

```
8       Accuracy: 0.9434
9       Precision: 0.9444
10      Recall: 0.9474
11      F1 Score: 0.9411
12      Precision, recall & F1: macro-averaged (equally weighted avg. of 3 classes)
13      ========================================================================
```

流式配置构造器允许我们很快的添加或修改神经网络的层级，或则调整一些参数从而验证我们的模型是否有所改善。

## 7．结论

通过这篇文章，我们使用 deeplearning4j 建立一个简单但是强大的神经网络。
跟以前一样，本文的源代码可以在 GitHub 中找到。