

ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-dl4j

dl4j

Peter Reutemann

December 20, 2017

©2016-2017



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-sa/4.0/>

Contents

1	Introduction	5
2	Flow	7
2.1	Iteration listeners	8
3	CUDA Support	9
4	OpenBLAS	11
5	Intel MKL	13
6	Off-heap memory	15
	Bibliography	17

Chapter 1

Introduction

The *dl4j* module makes the deeplearning4j[3] library available within ADAMS.

Due to the complex nature of configuring deep belief networks and such multi-layered neural networks, there is no graphical support for configuring networks as such. Instead, a so-called *model configurator* is used to return a configured model. This can be achieved in two ways:

- *Custom Java code* – you can subclass *AbstractModelConfigurator* (located in package *adams.ml.dl4j.model*) and implement your own parameters that are necessary for tweaking your network. This is done by the *SimpleMultiLayerNetwork* class in that package, which is just a configurable version of the network describe by deeplearning4j’s tutorial on the iris dataset.
- *Scripting* – taking advantage of either Groovy or Jython, you can use the *ModelWithScriptedConfiguration* class pointing to your script file that generates the actual network to be trained and used.

Chapter 2

Flow

The following sources are available:

- *DL4JDatasetIterator* – outputs datasets using the specified dataset iterator.
- *DL4JModelConfigurator* – defines a deeplearning model.
- *DL4JModelGenerator* – generates model(s) using the specified generator scheme.

The following transformers are available:

- *DL4JCrossValidationEvaluator* – cross-validates a model on the incoming dataset.
- *DL4JCrossValidationSplit* – generates sequence of train/test splits from the incoming dataset.
- *DL4JDatasetAppend* – combines multiple datasets into a single one, one after the other.
- *DL4JDatasetInfo* – outputs information about a dataset.
- *DL4JEvaluationSummary* – generates a summary from an Evaluation object/container.
- *DL4JEvaluationValues* – retrieves statistics from an Evaluation object/container and places them in a spreadsheet.
- *DL4JModelReader* – reads a serialized model from disk.
- *DL4JRandomizeDataset* – randomizes the incoming dataset.
- *DL4JRandomSplit* – generates a random split on the incoming dataset.
- *DL4JScoring* – generates scores (= predictions) for a dataset using a built model.
- *DL4JTestSetEvaluator* – evaluates an incoming built model on a callable dataset.
- *DL4JTrainModel* – builds a model obtained from a callable model configurator on the incoming dataset.
- *DL4JTrainTestSetEvaluator* – trains and evaluates a model obtained from a callable model configurator on the incoming train/test set container.

The following sinks are available:

- *DL4JModelWriter* – saves a model to disk using serialization.

The following conversions are available:

- *DL4JConfiguratorToModel* – obtains an actual model from a configurator.
- *DL4JDataSetToSpreadSheet* – turns a DL4J DataSet into a SpreadSheet.
- *DL4JJsonToModel* – turns a JSON[6] object into a model.
- *DL4JModelToJson* – turns a model into its JSON[6] representation.
- *DL4JModelToSpreadSheet* – turns the parameters of a model into a spreadsheet.
- *DL4JModelToString* – turns the parameters of a model into a simple string representation.
- *DL4JModelToJson* – turns a model into its JSON[6] representation.
- *DL4JModelToYaml* – turns a model into its YAML[7] representation.
- *DL4JYamlToModel* – turns a YAML[7] string into a model.
- *NDArrayToSpreadSheet* – turns a NDArray into a spreadsheet.
- *SpreadSheetToDL4JDataSet* – turns a spreadsheet into a DL4J DataSet.

2.1 Iteration listeners

By default, building models does not provide any insight into how the building is going. However, DL4J offers iteration listeners that can shed some light on the build process.

These listeners can be attached to the following actors:

- *DL4JCrossValidationEvaluator*
- *DL4JTrainModel*
- *DL4JTrainTestSetEvaluator*

The following schemes for configuring iteration listeners are available:

- *NullListener* – does nothing
- *CallableActorScoreListenerConfigurator* – forwards double array of iteration/score to the specified callable actor (e.g., for plotting)
- *ScoreIterationListenerConfigurator* – simple logging of statistics in console

Chapter 3

CUDA Support

By default, dl4j (actually nd4j[2]) uses the CPU for linear algebra. However, if you have CUDA installed, you can switch to GPU computation by adding another dependency in your *pom.xml* file.

The current version of nd4j supports CUDA[5] versions 7.5 and 8.0 at the time of writing¹, and requires you to specify which one you have. For example, if you have CUDA v7.5 installed, then you need to define the *artifactId* like this:

```
<dependency>
  <groupId>org.nd4j</groupId>
  <artifactId>nd4j-cuda-7.5</artifactId>
  <version>${nd4j.version}</version>
</dependency>
```

If you do not compile your own project, but simply use ADAMS releases/snapshots, then you can simply download the appropriate version of the CUDA libraries. E.g., for CUDA 7.5, this is *adams-dl4j-cuda-7.5-libs*. Just add all the files in the *lib* directory of this archive to the one of your ADAMS installation to enable CUDA support. But be aware that ADAMS will only work if you have CUDA support on the machine that you are running it on (a design drawback of deeplearning4j, unfortunately).

¹Check Maven Central for updated artifacts using <http://search.maven.org/#search%7Cga%7C1%7Cnd4j-cuda>

Chapter 4

OpenBLAS

By default, OpenBLAS determines automatically how many threads to use for speeding up the computing. However, you can use the following environment variable to manually set the number of threads¹:

`OMP_NUM_THREADS`

NB: This should be the number of *actual* cpus/cores. Systems with hyper-threading support will report multiples of this number.

¹<https://deeplearning4j.org/native>

Chapter 5

Intel MKL

Intel provides high-performance BLAS libraries for download as well[8]:

After installation, you have to add the libraries to your environment variables for deeplearning4j to pick up.

First, locate the directory that contains the MKL runtime library:

- Linux/OSX: `libmkl_rt.so`
- Windows: `mkl_rt.dll`

Second, add this directory to your environment variables:

- Linux/OSX: add the path to `LD_LIBRARY_PATH`

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/mkl_rt
```

- Windows: add the path to your `%PATH%` variable, either through the control panel or on the command prompt:

```
set PATH=%PATH%;path\to\mkl_rt
```

Furthermore, for Windows you also need to add the OpenMP runtime libraries to the path. Locate the directory that contains the `libiomp5md.dll` library and add this directory to your `%PATH%` environment variable as well.

Chapter 6

Off-heap memory

dl4j uses JavaCPP's feature for off-heap memory usage. By default, this uses a maximum size that is twice Java's physical memory size. However, when running large networks, this heuristic may be not enough, as the Java side of things may not actually require a lot of memory, but the off-heap one.

In order to manually set the off-heap maximum memory, you can use the following JVM parameter on the command-line to set a limit of 20GB:

```
-Dorg.bytedeco.javacpp.maxphysicalbytes=20000000000
```

When using the launcher script or one of the other ADAMS supplied scripts, you can use the following command-line to pass the parameter to the JVM:

```
-jvm -Dorg.bytedeco.javacpp.maxphysicalbytes=20000000000
```


Bibliography

- [1] *ADAMS* – Advanced Data mining and Machine learning System
<https://adams.cms.waikato.ac.nz/>
- [2] *nd4j* – N-Dimensional Arrays for Java.
<http://deeplearning4j.org/>
- [3] *deeplearning4j* – open-source, distributed, deep-learning library for the JVM.
<http://nd4j.org/>
- [4] *JCuda* – Java bindings for CUDA.
<http://jcuda.org/>
- [5] *CUDA* – a parallel computing platform and programming model invented by NVIDIA.
http://www.nvidia.com/object/cuda_home_new.html
- [6] *JSON* – JavaScript Object Notation is a lightweight data-interchange format
<http://json.org/>
- [7] *YAML* – is a human friendly data serialization standard for all programming languages.
<http://yaml.org/>
- [8] *MKL* – Math Kernel Library
<https://software.intel.com/en-us/mkl>