

ADAMS

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

Module: adams-dl4j

dl4j

Peter Reutemann

December 21, 2016

©2016



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
3	CUDA Support	9
	Bibliography	11

Chapter 1

Introduction

The *dl4j* module makes the deeplearning4j[2] 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.

The following transformers are available:

- *DL4JRandomSplit* – generates a random split on the incoming dataset.
- *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.
- *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:

- *DL4JModelToJson* – turns a model into its JSON[6] representation.
- *DL4JModelToYaml* – turns a model into its YAML[7] representation.
- *NDArrayToSpreadSheet* – turns a NDArray into a spreadsheet.

Chapter 3

CUDA Support

By default, dl4j (actually nd4j[3]) 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 JCuda[4] supports the CUDA[5] versions 5.5, 6.0, 6.5 and 7, and requires you to specify which one you have. For example, if you have CUDA v6.0 installed, then you need to define the *artifactId* like this:

```
<dependency>
  <groupId>org.nd4j</groupId>
  <artifactId>nd4j-cuda-6.0</artifactId>
  <version>${nd4j.version}</version>
</dependency>
```


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/>