

---

# Table of Contents

Introduction	1.1
Simplest Network Lab	1.2
DataVec Lab	2.1
FeedForward Classification Lab	3.1
LSTM Lab	4.1
Building a Convolutional Neural Network	4.2
Model Saving and Loading Lab	4.3

# Introduction

The Labs for this course are completed using IntelliJ. Your instructor will have provided either instructions for setting up IntelliJ or you will have a Virtual Machine with IntelliJ pre-installed and configured.

## 1. Maven

We recommend you familiarize yourself with at least the basics of Maven for managing your DeepLearning4J projects.

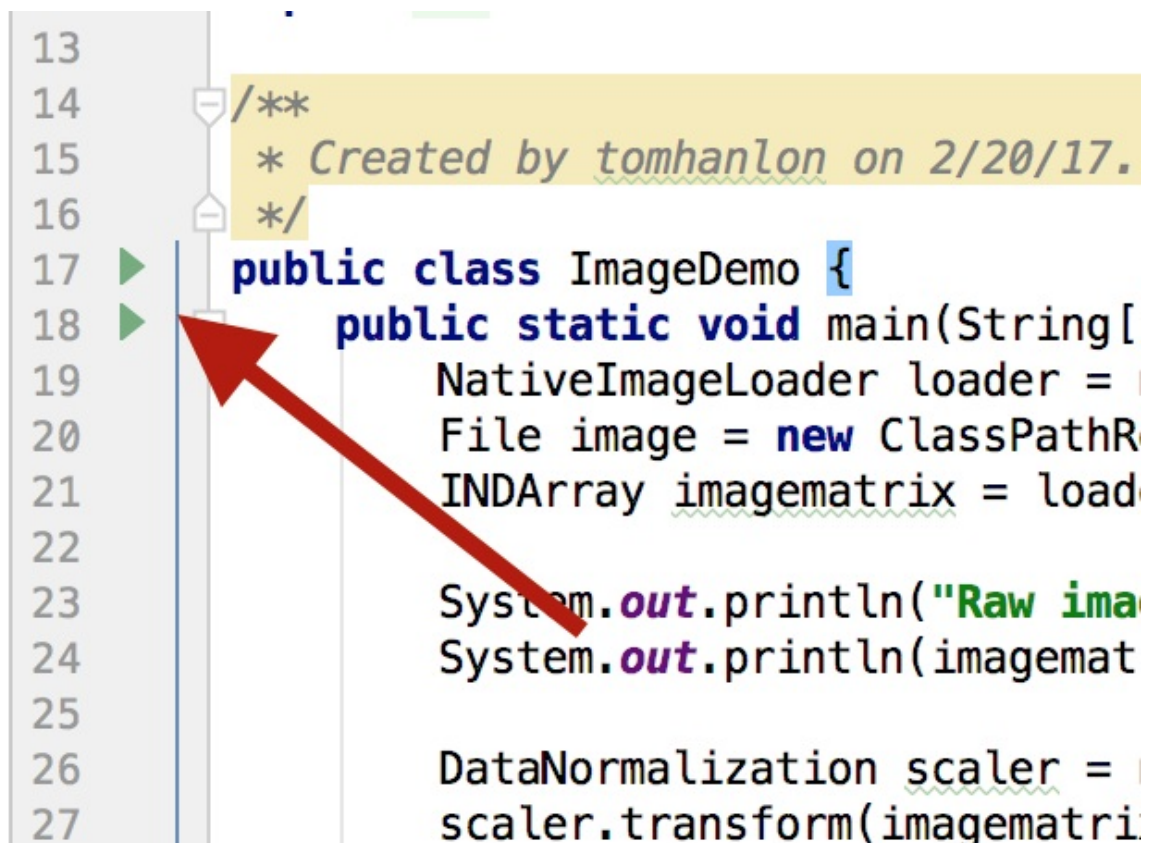
Maven uses a pom.xml file to manage dependencies.

If exploring the pom.xml file for the lab project note that there are two levels, training-parent/pom.xml and training-parent/training-labs/pom.xml

## 1. Running and stopping code

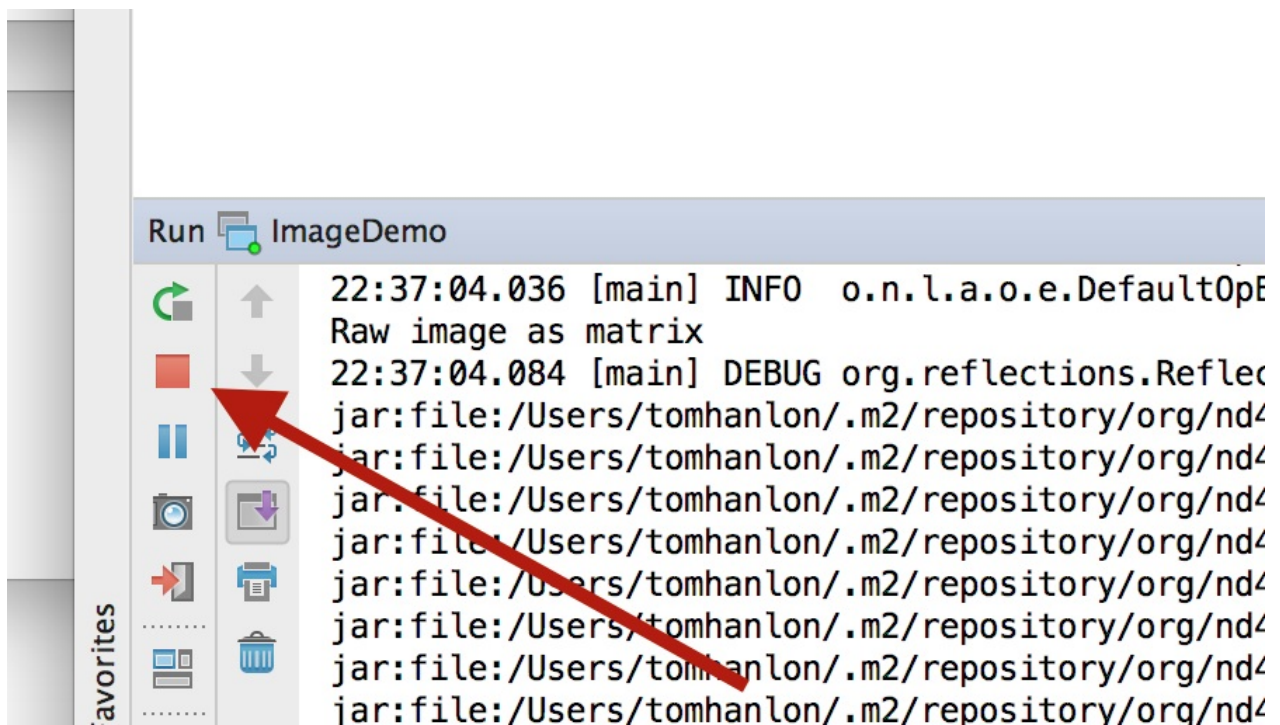
When editing or creating a java class you can run it at any time by hitting the green arrow next to your main class.

- Run your code



- Stop your code

To stop the code you can hit the red button.



### 1. Common issues

A code problem anywhere is a problem everywhere. If the project fails to compile then you will not be able to run other classes that do compile. Running a class causes the whole project to compile, and if another broken class fails to compile, your working class will fail to run.

Solution, if you have an unfinished class or something that won't compile, you need to either fix it, or comment out the offending section

1. Work at your own pace.
2. Too Hard

There is a solution directory, use it as a last resort. If the Lab is too challenging remember you can work as far as you can, save the class in a working state or comment it out to the point where it at least compiles and move on.

- Too Easy

You are welcome to start a blank project and work from scratch. If your questions are related to the course materials or the topic the instructor will happily answer your questions

# Simplest Neural Network

In this Lab you will explore a very simple Neural Network. The Neural Network will consist of

- One input
- One expected numeric output
- One Hidden Layer with a single neuron

The Neural Net will receive the input of 0.5, the expected output will be 0.8 .

The Neural Net will train for 100 epochs with the goal of improving it's score, how close it gets to 0.8 .

## Goals of this lab

- To familiarize the user with DeepLEarning4J code.
  - MultiLayer Network
    - fit
  - Parameters
    - Number of Epochs
    - Training Rate
    - Optimization Algorithm
    - Updater
  - UI server

## Step 1

- Open up IntelliJ Open up IntelliJ and navigate to the Labs folder

## Step 2

- Open the SimplestNetwork class

Click on SimplestNetwork.java to open up the java class in the editor

## Step 3

- Review the Java Code

Note the parameters set at the top.

```
int seed = 123;
```

This is a hardcoded random seed to allow repeatable results. The Neural Net begins by assigning random weights to the matrix(?). If we want repeatable results then using a pre configured seed allows that.

If you change the seed, your networks behavior will change slightly as well.

```
int numInputs = 1;  
int numOutputs = 1;
```

## Xavier, why Xavier

In short, it helps signals reach deep into the network.

If the weights in a network start too small, then the signal shrinks as it passes through each layer until it's too tiny to be useful. If the weights in a network start too large, then the signal grows as it passes through each layer until it's too massive to be useful. Xavier initialization makes sure the weights are 'just right', keeping the signal in a reasonable range of values through many layers.

To go any further than this, you're going to need a small amount of statistics - specifically you need to know about random distributions and their variance.

## STOCHASTIC GRADIENT DESCENT

Note that the Optimiaztion Algorithm is Stochastic Gradient Descent.

As Neural Netowrks have been researched over the years the challenge of updating large matrices with modified weights to lead to less error(better answers) has been significant. The numerical computation in particular. SGD meets this challenge by making random choices in some way, research this further.

## Updater Nesterovs

Without going into the updater in detail Note that momentum may be a hyperparameter that will need tuning on more complex networks. The problem in this demo is linear (? is it) but in a more complex graph with potential local minima momentum helps break through that. How deep do I go here?

## Layer 0 activation tanh

The activation function of a Layer determines the signal it sends to connected neurons.

Choices are sigmoid, smooth curve output 0-1 as x increases.

tanh similar to sigmoid output -1-> +1 depending on value of x

Stepwise output 0 or 1 depending on value of X

Etc going to deep here.

## Layer1 this is our output layer.

Note the activation is identity.

This determines that the output will be linear, a range of numeric values, .1, .2, .3 etc.  
VS 0 or 1, vs Class A, B or C

## STEP 3

Run the code

In this step you will run the code.

When the code executes it will create a UI that can be accessed with a web browser.

It will also print output to the output window at the bottom of intellij as it runs

**Click on this green arrow to execute the code**

```
22
23
24 * Built for SkyMind Training class
25 */
26 public class SimplestNetwork {
27     private static Logger logger = LoggerFactory.getLogger(SimplestNetwork.class);
28     public static void main(String[] args) throws Exception{
29         /*
30          * Most Basic NN that takes a single input
31          */
32
```

## View the output in the console while the class runs

```
52
53
54 INDArray input = Nd4j.create(new float[] {(float) 0.5}, new int[] {1,1}); // Our input value
55 INDArray output = Nd4j.create(new float[] {(float) 0.8}, new int[] {1,1}); // expected output
56
```

Run: SimplestNetwork

```
o.d.o.l.ScoreIterationListener - Score at iteration 8 is 2.1712028980255127
o.d.e.d.SimplestNetwork - -0.64
o.d.o.l.ScoreIterationListener - Score at iteration 9 is 2.06229829788208
o.d.e.d.SimplestNetwork - -0.60
o.d.o.l.ScoreIterationListener - Score at iteration 10 is 1.952489972114563
o.d.e.d.SimplestNetwork - -0.56
o.d.o.l.ScoreIterationListener - Score at iteration 11 is 1.8429129123687744
o.d.e.d.SimplestNetwork - -0.52
o.d.o.l.ScoreIterationListener - Score at iteration 12 is 1.7345409393310547
o.d.e.d.SimplestNetwork - -0.48
o.d.o.l.ScoreIterationListener - Score at iteration 13 is 1.6281943321228027
o.d.e.d.SimplestNetwork - -0.43
```

## View the UI

When the code executes and the UI is created, a line is generated in the console output with the url

```
55
56 INDArray output = Nd4j.create(new float[] {(float) 0.8}, new int[] {1,1}); // expected output
57
```

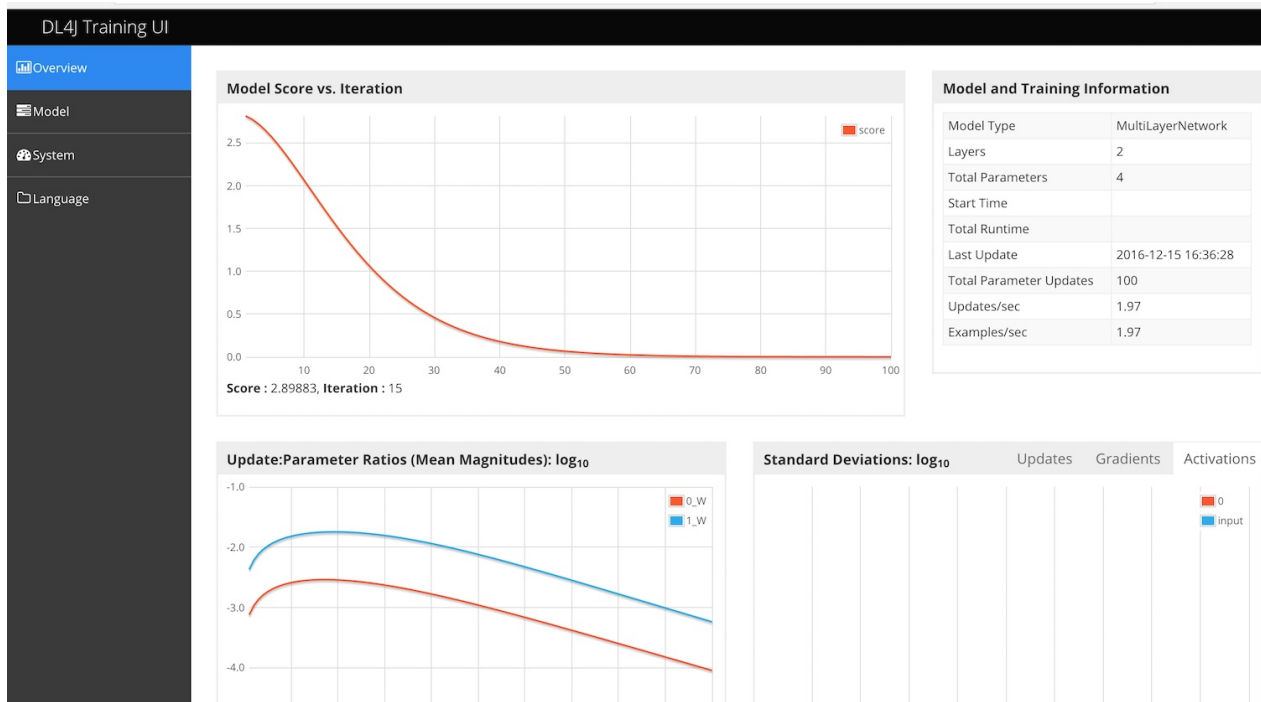
Run: SimplestNetwork

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_77.jdk/Contents/Home/bin/java ...
o.n.n.NativeOps - Number of threads used for NativeOps: 4
Unable to guess runtime. Please set OMP_NUM_THREADS or equivalent manually.
o.n.n.Nd4jBlas - Number of threads used for BLAS: 4
o.d.e.d.SimplestNetwork - 0.50
o.d.n.c.MultiLayerConfiguration - Warning: new network default sets pretrain to false.
o.d.n.c.MultiLayerConfiguration - Warning: new network default sets backprop to true.
o.d.u.p.PlayUIServer - UI Server started at http://localhost:9000
o.d.u.p.PlayUIServer - StatsStorage instance attached to UI: InMemoryStatsStorage(uid=91e4832d)
o.d.o.l.ScoreIterationListener - Score at iteration 0 is 2.8157103061676025
o.d.e.d.SimplestNetwork - -0.87
```

Open that URL in a browser

You should see this





## Explanation of the output

Console Output.

The following block of code is what begins the training process.

```
for( int i=0; i<nEpochs; i++ ){
    model.fit(input,output);
    INDArray output2 = model.output(input);
    log.info(output2.toString());
    Thread.sleep(500);
}
```

## What is an Epoch?

It is a loop for the total number of Epochs. Or total passes through the training dataset, in this case our single input, but in real use cases it might be something like thousands of text reviews, or hundreds of thousands of images, or millions of lines from log files.

## What is Model.fit?

This is where the model trains. Data is ingested, random weights are assigned, output is evaluated against expected and weights are adjusted to lessen the error.

# What output should look like

This section

```
INDArray output2 = model.output(input);  
log.info(output2.toString());
```

Generates these lines in our console output.

```
o.d.e.d.SimplestNetwork - -0.87  
o.d.e.d.SimplestNetwork - -0.85
```

The "correct" output, or "expected" output is 0.80, you will see that the network is consistently getting closer to that goal as it trains.

This line in the console output

```
o.d.o.l.ScoreIterationListener - Score at iteration 1 is 2.775866985321045
```

Is generated by this line

```
model.setListeners(new StatsListener(statsStorage), new ScoreIterationListener(1));
```

## STEP 4

In this step you will modify some of the parameters and see the effect on the training process.

Note that anytime you re-run this code you will have to terminate the previous running process. The webserver serving the UI will have a handle on a socket and the second example will try to grab that same socket, fail and return an error.

Kill the running process by clickin on the red square, top right.



Some parameters that you could tune.

Before you change things, note the current performance. How many iterations till it got to within .05 of the target? In 100 iterations how close did it get? Mine got to .78 after 100, and reached .75 at iteration 80

## Settings you may change with reasonable results

### Hidden Nodes

- Number of hidden nodes
  - Would provide more attempts towards the correct answer, more random weights, and may train quicker

### Number of Epochs

- Number of Epochs
  - If the network is converging on the target, then more epochs should allow it to get there, in time
  - Note that to prevent things going to fast to visualize, I put a sleep .5 seconds in the loop.
  - Remove that if you set to large number of Epochs.

## Learning Rate

Learning Rate Determines how far to adjust the weights given the error.

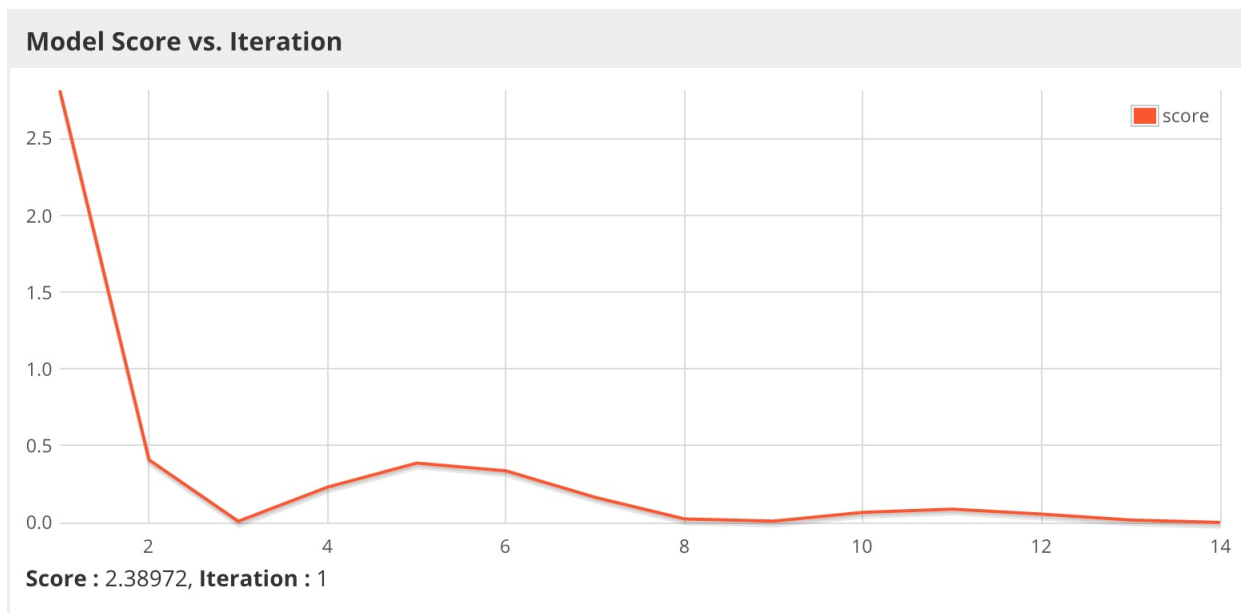
A range for learning rates would be ???

```
double learningRate = 0.001;
```

Change to perhaps...

```
double learningRate = 0.01;
```

Note that an aggressive learning rate may cause the network to overshoot the target before converging.



## Lab questions

1. What parameters may need adjusting in a Neural Net

# END OF SIMPLEST NETWORK LAB

# Using DataVec to ingest a CSV dataset

In this lab you will import data from a CSV file into a format suitable for a Neural Network.

## Goals of this lab

- DataVec Introduction

## Step 1

- Open up IntelliJ Open up IntelliJ and navigate to the Labs folder

## Step 2

- Open the DataVecLab class

Click on DataVecLab.java to open up the java class in the editor

## Step 3

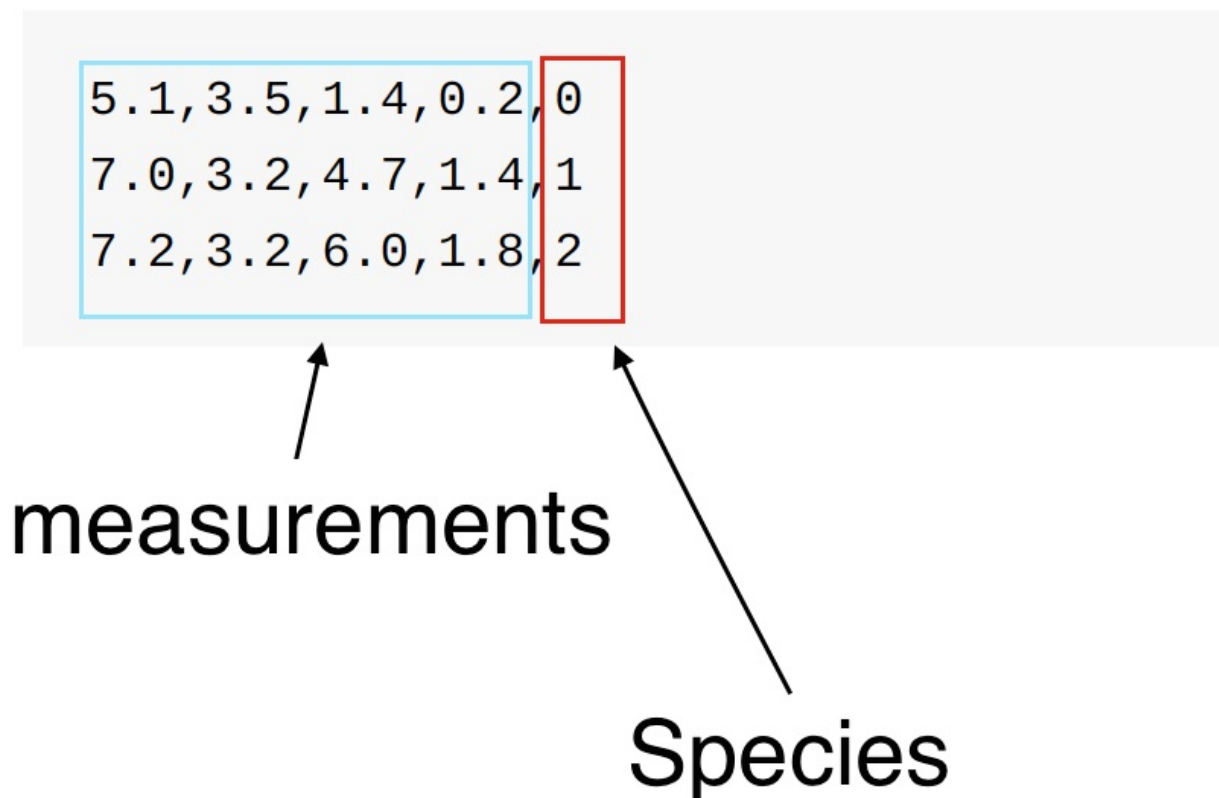
- Review the Java Code

In this case the Neural Network has already been built. The goal of this Lab is to work through the Data ETL process using DataVec.

- Understand the Challenge

The Iris.txt file has 150 records of measurements of 3 Irises. Iris Setosa, Iris Virginica, Iris versicolor. Measurements are Petal Length, Petal Width, Sepal Length, Sepal Width.

The data is stored with numeric representation of the species, 0=> Setosa, 1 => Versicolor, 2=> Virginica



## STEP 3

Review the needed steps.

1. Read the File
2. Parse the Lines
3. Specify Label fields vs measurements
4. Create a DataSet object to pass into our Neural Network.

DataVec Classes that will be used.

<https://deeplearning4j.org/datavecdoc/org/datavec/api/records/reader/RecordReader.html>

<https://deeplearning4j.org/datavecdoc/org/datavec/api/records/reader/impl/csv/CSVRecordReader.html>

DeepLearning4J class that will be used

<https://deeplearning4j.org/doc/org/deeplearning4j/datasets/datavec/RecordReaderDataSetIterator.html>

Full DataVec JavaDoc <https://deeplearning4j.org/datavecdoc/>

Full DeepLearning4J JavaDoc. <https://deeplearning4j.org/doc/>

Advanced users are welcome at this point to open up the stub and go for it.

Everyone else please follow along with the instructions

## STEP 3 Set some parameters

CSVRecordReader is designed to be able to ignore the first x number of lines in a file. The assumption is the file may have header information or comments.

Take a look at Iris.txt and confirm that it has no headers.

CSVRecordReader is configurable in terms of how the data records are delimited. Verify that the file is comma delimited.

**\*\* Note** bad data is a frequent problem, in this clean sterilized lab environment you can trust the data, in real world I always run some verification scripts to verify every line has the same amount of commas, at the very least.

After verifying that there are no header lines, and the delimiter is a comma, add the following code to the stub program.

```
int numLinesToSkip = 0;  
String delimiter = ",";
```

## Create a RecordReader

Add this line to the code stub

```
RecordReader recordReader = new CSVRecordReader(numLinesToSkip, delimiter);
```

## Initialize the RecordReader and pass it a file.

For portability the file is put in the resources folder. This makes it available as a `ClassPathResource`. If you chose to get the path to the file and use that instead that is fine.

This code snippet allows easy access to a file on the classpath `new`  
`ClassPathResource(fileName).getFile()`

Initialize your record Reader and passed it a `FileSplit`.

A `FileSplit` can point to a directory and the Record Reader can read all the files in the directory, or in this case it will point to a single file.

```
recordReader.initialize(new FileSplit(new ClassPathResource("iris.txt").getFile())
);
```

## Optional Step

Verify the Record Reader.

You may want to verify the the Record Reader code is functional.

The Record Reader returns an Iterator over a set of Records.

Each call to next method returns an `java.util.ArrayList` of values.

Some code to explore that would be.

```
while( recordReader.hasNext()) {
    log.info(recordReader.next().toString());
    log.info(recordReader.next().getClass().toString());
}
recordReader.reset();
```

You would then run the code and see the records in the console window.

## Set some parameters for the DataSetIterator that you will create in the next step



```
int labelIndex = 4;
//5 values in each row of the iris.txt CSV:
//4 input features followed by an integer label (class) index.
// Labels are the 5th value (index 4) in each row

int numClasses = 3;
//3 classes (types of iris flowers) in the iris data set.
//Classes have integer values 0, 1 or 2

int batchSize = 150;
//Iris data set: 150 examples total.
//We are loading all of them into one DataSet
//(not recommended for large data sets)
```

## Create a DataSet Iterator

A Record Reader returns an iterator over a List of Writables. Writables are an efficient serialization method inspired by Hadoop Writables.

A Neural Net requires input as an Array of Numeric values. To do that use DataSetIterator.

A DataSet will contain an INArray of features, and an INArray of Labels.

Add the following code to the DataVecLab class.

```
DataSetIterator iterator = new RecordReaderDataSetIterator(recordReader, batchSize,
labelIndex, numClasses);
DataSet allData = iterator.next();
```

## Shuffle the Data

Typically a Neural network trains over the results of a minibatch. Suppose a minibatch of Ten. Ten records are passed through the error between expected value and observed output is calculated and the weights of the network are updated to reduce the error. Now look at our iris data set. All one species, followed by another species. If the records in a mini-batch are skewed towards one class then the network will train first in one direction then another. This is not good, shuffle your data.

Add this code to the class.

```
allData.shuffle();
```

## Split Train and Test

For supervised learning a network is trained on a collection of records and then tested on records it has not seen before. Split the data into test and train.

Add the following code to the DataVecLab class.

```
SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.65); //Use 65% o
f data for training
DataSet trainingData = testAndTrain.getTrain();
DataSet testData = testAndTrain.getTest();
```

## Extra Credit

Normalize the DataSet

You should always Normalize/standardize your data. DeepLearning4J has tools to do that.

<http://nd4j.org/doc/org/nd4j/linalg/dataset/api/preprocessor/DataNormalization.html>

## Extra Credit

Your data may be organized in many different ways. See the resources/irisData directory for some examples.

### Parent Path Label Generator:

Labels can be extracted based on the name of the parent directory using ParentPathLabelGenerator. The irisData directory has 3 folders one for iris virginica, one for iris setosa and one for iris versicolor.

Write a datavec pipeline that extracts the label from the parent directory.

## DataVec Spark Transform

Your data may have string labels instead of numeric values. DataVec has tools to build a Schema as read from the file, and then a target schema that transforms columns. In this case you would want to transform categorical to integer.

An example of Spark Transform is available in demos/AbaloneDataTransform

In that demo the data has String Labels for Gender, M(male),F(Female),I(Infant)

A similar format for Iris Data is in the resources folder,

```
/resources/IrisData/iris_with_names.csv
```

## END OF IRIS DATAVEC LAB

# Lab: Using a FeedForward Network as Classifier

- Abalone data
    - Gender
    - Weight
    - Height etc
  - Question for the Neural Network
    - Given that data can we predict age
    - Otherwise age is calculated by killing the abalone and counting rings
-

# Data Source

- University of California at Irvine
  - Large collection of DataSets
    - <http://archive.ics.uci.edu/ml/datasets.html>
-

# What is an Abalone



Copyright: The original uploader was Geographer at English Wikipedia - Transferred from en.wikipedia to Commons., CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=4384795>

---

## More about Abalone

- Tasty Seafood
  - High harvest demand needs to be managed
  - Populations are studied
  - Therefore we have a dataset
-

# Dataset Details

- Total Records 4177
    - Not a lot of data
    - Question: How much Data do I need?
    - Answer: More, is always better
  - Data structure
-



## DataSet Details

- Sex / nominal / -- / M, F, and I (infant)
  - Length / continuous / mm / Longest shell measurement
  - Diameter / continuous / mm / perpendicular to length
  - Height / continuous / mm / with meat in shell
  - Whole weight / continuous / grams / whole abalone
  - Shucked weight / continuous / grams / weight of meat
  - Viscera weight / continuous / grams / gut weight (after bleeding)
  - Shell weight / continuous / grams / after being dried
  - Rings / integer / -- / +1.5 gives the age in years
-

## Specifics

- DataSet has number of rings, not age
  - Rings +1.5 gives age
  - More straightforward to predict rings, extrapolate age
-

# Loading our data

- Gender M/F/I, first column, needs to be converted to numeric
  - Last column is our label, number of rings
  - All other columns doubles
-

# Data considerations

- Better performance when values normalized/scaled between 0 and 1
  - Are our values normalized?
  - Run then through datavec to get statistics
  - Also convert string to categorical

***note the data has been prepared and split, the raw data and the process are available in resources folder***

---

# Analyzing the Data

- See Demos/AbaloneDataTransform.java
  - Loads data
  - Transforms M/F/I to 0,1,2
  - Uses Spark RDD's for transform (RDD Docs)[  
<http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds>]
  - You can run that code, it places summaries in /tmp/Abalonexxx
-

# STEP 1

- Run the code in Demos/AbaloneDataTransform.java
-

## STEP 2

- Examine the files in /tmp/abalone\_data\_XXXX
    - the XXXX will be replaced with the minute of the day to allow the code to run more than once without error
  - Examine /tmp/abalone\_data\_XXXX/analysis.tx
    - Is the data normalized, are the values between 0-1?
  - Examine the directory /tmp/abalone\_data\_XXXX/original/
    - Question: Why are there two files?
    - Answer: Spark is distributed by design, although it is run locally here it still has two process each on a split of the data. You can not have two processes write to same file, so spark/hadoop output is a directory containing maining files.
  - Examine a part file: "less /tmp/abalone\_data\_XXX/original/part-00000"
    - Is the data Numeric?
  - Examine a part file: "less /tmp/abalone\_data\_XXX/processed/part-00000"
    - Is the data Numeric?
-

## STEP 3

In this step you will begin writing code to process the data. The data has been converted into numeric values for the Gender column, and split into Train (resources/abalone/abalone\_train.csv) and Test (resources/abalone/abalone\_test.csv) sets.

- Create File objects traindata and testdata

```
File traindata = new ClassPathResource("abalone/abalone_train.csv").getFile();
File testdata = new ClassPathResource("abalone/abalone_test.csv").getFile();
```

## STEP 4

- Create a Recordreader for the Test data and one for the Train data.
- Initialize each
- You will need four lines of code.

## STEP 5

- Create a DataSetIterator for each RecordReader, remember to use the constructor with batchSize, LabelIndex and the other needed fields.

## STEP 6

- Uncomment the code for the Neural Network

## STEP 7

- Uncomment the code to run the model



## Final Notes

- This Data does not give enough information for particularly accurate age predictions. Too much variability in growth patterns. Take a look at the output and it gets within a few years for most.
- The Neural Net could most likely be improved and results improved but this is real data and the correlation between age and the measurements provided is present, but not particularly strong.

## Extra Credit

### Larger ranges for years

- Divide the data in another way, perhaps age 0-5, 5-10, 10->maxvalue, and see how well the measurements can target the broader age groups.
  - Hint: Note the relationship between number of inputs, number of outputs and number of classes when you make this change

### Single continuous output

- Reconfigure the network to have a single output node delivering a continuous value. Instead of age = 1 of class 0-30, have age = continuous value 0-30
- Loss Function
  - For the regression output layer loss function we have multiple valid choices.
  - The most common are mean squared error (MSE) or sum of squared errors (L2)
- Activation Function
  - In this case we use the identity (linear) output function

**END OF ABALONE LAB**



# Character Generation LSTM Lab

In this lab you will work on a Java Class that implements an LSTM Recurrent Neural Network. Long Short Term Memory Recurrent Networks are modelled after the work done by Graves (note source). Unlike a simple Multi Layer Perceptron, the computation nodes of an LSTM have the ability to recognize patterns in time series data. Useful for many Time Series applications, this lab treats a weather forecast as a sequential series of characters and predicts the next character.

## Goals of this lab

- To familiarize the user with LSTM network configuration and use.

## Lab Overview

This Lab will train a Neural Network to generate weather forecasts.

The Weather Forecast training data is gathered from actual National Weather service station Forecast Products.

Here is a sample forecast product.

WVZ015-171700-  
KANAWHA-  
INCLUDING THE CITIES OF...CHARLESTON...SOUTH CHARLESTON...  
SAINT ALBANS  
932 PM EST FRI DEC 16 2016

.REST OF TONIGHT...CLOUDY. RAIN LIKELY. NOT AS COLD WITH LOWS AROUND  
30. TEMPERATURE RISING INTO THE LOWER 40S. SOUTH WINDS 10 TO 15 MPH  
WITH GUSTS UP TO 25 MPH. CHANCE OF RAIN 70 PERCENT.  
.SATURDAY...CLOUDY. RAIN LIKELY IN THE MORNING...THEN A CHANCE OF  
SHOWERS IN THE AFTERNOON. MUCH WARMER WITH HIGHS IN THE LOWER 60S.  
SOUTHWEST WINDS 10 TO 15 MPH WITH GUSTS UP TO 25 MPH. CHANCE OF RAIN  
70 PERCENT.  
.SATURDAY NIGHT...SHOWERS...MAINLY AFTER MIDNIGHT. LOWS IN THE UPPER  
30S. SOUTHWEST WINDS 10 TO 15 MPH. CHANCE OF RAIN NEAR 100 PERCENT.  
.SUNDAY...RAIN SHOWERS IN THE MORNING...THEN SNOW SHOWERS LIKELY IN  
THE AFTERNOON. MUCH COOLER WITH HIGHS IN THE LOWER 40S. TEMPERATURE  
FALLING TO AROUND 30 IN THE AFTERNOON. WEST WINDS 5 TO 10 MPH.  
CHANCE OF PRECIPITATION 90 PERCENT.  
.SUNDAY NIGHT...MOSTLY CLOUDY. A SLIGHT CHANCE OF SNOW SHOWERS IN  
THE EVENING. MUCH COLDER WITH LOWS IN THE LOWER 20S. NORTHWEST WINDS  
5 TO 10 MPH. CHANCE OF SNOW 20 PERCENT.  
.MONDAY AND MONDAY NIGHT...PARTLY CLOUDY. HIGHS IN THE LOWER 30S.  
LOWS IN THE LOWER 20S.  
.TUESDAY THROUGH WEDNESDAY...MOSTLY CLEAR. HIGHS IN THE MID 40S.  
LOWS IN THE MID 20S.  
.WEDNESDAY NIGHT...PARTLY CLOUDY IN THE EVENING...THEN BECOMING  
MOSTLY CLOUDY. LOWS IN THE MID 30S.  
.THURSDAY AND THURSDAY NIGHT...MOSTLY CLOUDY. A CHANCE OF RAIN  
SHOWERS. HIGHS IN THE MID 40S. LOWS IN THE LOWER 30S. CHANCE OF RAIN  
40 PERCENT.  
.FRIDAY...MOSTLY SUNNY. HIGHS IN THE MID 40S.

\$\$

## Details worth noting

The Text is not exactly proper english. Looks like "." at beginning of line denotes beginning of phrase, no "." at beginning of line denotes continuation of line, "\$\$" denotes end of product.

## How our net will receive the data

The data will be delivered as a sequence of single characters. It will attempt to predict the next character, evaluate the results update the weights and repeat.

It will begin to learn to sound sort of like a weather forecast.

The output will be started with this sequence.

```
WVZ006-171700-  
CABELL -  
INCLUDING THE CITY OF...HUNTINGTON  
932 PM EST FRI DEC 16 2016
```

And the neural net will generate the rest

Here is a sample of a first attempt.

```
.TODAY...MOSTLY CLOUDY. A CHANCE OF RAIN. NOT0 10 TO 16. NOUTHWEST WINDS 30 TOR 10  
0. LOWS AROUND 30. WEST WINDS AROUND  
5 MPH. CHANCE OF RAIN 60 PERCENT.  
.SATURDAY...CLOUDY. HIGHS IN THE LOWER 30S. CHANCE OF PRECIPITATION 60 PERCENT.  
.SUNDAY NIGHT...MOSTLY CLOUDY IN THE MORNING...THEN BECOMING  
MOSTLY  
CLOURNG WITH A 40 PERCENT CHANCE OF SHOW SHOWERS. COLD WITH HIGHS IN THE LOWER 40S  
.  
.FRIDAY NIGHT...MOSTLY CLOUDY. A SLIGHT CHANCE OF RAIN  
IN THE AFTERNOON.  
COOL WITH HIGHS IN THE MID 40S.
```

```
WVZ028-020150-  
WROING-  
INCLUDING THE CITY OF...GASTAY...CLEAN...BESPARAY HIGE 5GHT A  
COOL WITH HIGHS IN THE UPPER 40S. CHANCE OF SNOW 30 PERCENT.  
.SUNDAY NIGHT...A CHANCE OF SNOW. NOT AS COOL NH T A CHANCE OF SNOW SHOWERS. COLD  
WITH LOWS IN THE UPPENNOG...TUEN ANOVVEATUR HISH A MOND 10. NORTHWEST WINDS  
AROUND  
5 MPM.  
CHANCE OF R0E C0MPERATURE IN  
THE MID AST.  
.FRIDAY...CLOUDY. A SLIGHT CHANCE OF SNOW AND RAIN THIS LOWS IN THE AFTERNOON. MUC  
H COOLER. NEAR STE LID 30 PM.  
.THURSDAY...SUNNY. COLDER. NEAR STEADY TEMPERATURE IN THE  
LOWER 30S.  
.SUNDAY...MOSTLY SUNNY. HIGHS IN THE MIDNIGHT. COLD WITH HIGHS IN THE LOWER 20S.  
.SUNDAY NIGHT...RAIN. NOT AS COOL WITH LOWS IN THE MPR 30T. CHANCE OF PREC
```

Most of the output is words. Some not so much. Seems to have learned the beginning of line sequence. Will it ever learn the order of days?

## Step 1

- Open up IntelliJ Open up IntelliJ and navigate to the Labs folder

## Step 2

- Open the GravesLSTMCharModellingWeatherForecasts
- Open up the java class in the editor

## Step 3

- Review the Java Code

Note the parameters set at the top.

```
int lstmLayerSize = 200; //Number of units in each GravesLSTM layer
int miniBatchSize = 32; //Size of mini batch to use when training
int exampleLength = 4000; //Length of each training example sequence to use. This could certainly be increased
int tbpttLength = 50; //Length for truncated backpropagation through time. i.e., do parameter updates ever 50 characters
int numEpochs = 1; //Total number of training epochs
int generateSamplesEveryNMinibatches = 10; //How frequently to generate samples from the network? 1000 characters / 50 tbptt length: 20 parameter updates per minibatch
int nSamplesToGenerate = 4; //Number of samples to generate after each training epoch
int nCharactersToSample = 1200; //Length of each sample to generate
// String generationInitialization = null; //Optional character initialization; a random character is used if null
String generationInitialization = "WVZ006-171700-\n" +
    "CABELL-\n" +
    "INCLUDING THE CITY OF...HUNTINGTON\n" +
    "932 PM EST FRI DEC 16 2016";
// Above is Used to 'prime' the LSTM with a character sequence to continue/complete.
// Initialization characters must all be in CharacterIterator.getMinimalCharacterset() by default
Random rng = new Random(12345);
```

## Each Parameter Described

- `lstmLayerSize = 200`
  - Each of the lstm layers will have 200 nodes
- `miniBatchSize = 32;`
  - Number of characters
- `exampleLength = 4000;`
- `tbpttLength = 50;`
  - Parameter updates every 50 characters
- `generateSamplesEveryNMinibatches = 10;`
  - How often to generate a sample
- `nSamplesToGenerate = 4`
  - Number of samples to generate after each training epoch
- `nCharactersToSample = 1200;`
  - Length of each sample to generate
- `Random rng = new Random(12345);`

- Consistent Random Seed for consistent results

String generationInitialization

WVZ006-171700-CABELL- INCLUDING THE CITY OF...HUNTINGTON 932 PM EST  
FRI DEC 16 2016

## LAB STEP 1

- Set up your LSTM Neural Network

Create a MultiLayerNetworkConfiguration Object

Here is a starting Line, use the Builder method and add

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()  
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT).iterations(1)
```

Add a learning rate of(0.1) Set rmsDecay to 0.95 Set seed to 12345 Set regularization to true Set l2 0.001 Set weightInit to Xavier Set updater RMSPROP Add .list()

## Lab Step 2

Add 3 new Layers

```
.layer(0, new GravesLSTM.Builder().nIn(iter.inputColumns()).nOut(lstmLayerSize)  
    .activation(Activation.TANH).build())  
.layer(1, new GravesLSTM.Builder().nIn(lstmLayerSize).nOut(lstmLayerSize)  
    .activation("tanh").build())  
.layer(2, new RnnOutputLayer.Builder(LossFunction.MCXENT).activation("softmax")  
    .nIn(lstmLayerSize).nOut(nOut).build())
```

## Lab Step 3

Configure the BackPropagation settings for the Neural Network/



LSTM use Backprop through Time to update the weights. Remember in the slides the discussion of an RNN being a "stacked" FeedForward Network for each timestamp with constraints? In this case use `updateBackpropType.TruncatedBPTT`, use `tbpttLength` that is set to 50 for the forward and `backwardLength`

Set `Pretrain` to false, `pretrain` only applies to VAE and RBM's.

Set `backprop` to true.

```
.backpropType(BackpropType.TruncatedBPTT).tbPTTForwardLength(tbpttLength).tbPTTBackwardLength(tbpttLength)
    .pretrain(false).backprop(true)
    .build();
```

## Lab Step 4

Add the following code to create a MultiLayer Network from the configuration created above.

Initialize the Neural Net and attach a Listener so you can view progress

```
MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.init();
net.setListeners(new ScoreIterationListener(1));
```

## Xavier, why Xavier

In short, it helps signals reach deep into the network.

If the weights in a network start too small, then the signal shrinks as it passes through each layer until it's too tiny to be useful. If the weights in a network start too large, then the signal grows as it passes through each layer until it's too massive to be useful. Xavier initialization makes sure the weights are 'just right', keeping the signal in a reasonable range of values through many layers.

To go any further than this, you're going to need a small amount of statistics - specifically you need to know about random distributions and their variance.

## STOCHASTIC GRADIENT DESCENT

Note that the Optimization Algorithm is Stochastic Gradient Descent.

As Neural Networks have been researched over the years the challenge of updating large matrices with modified weights to lead to less error(better answers) has been significant. The numerical computation in particular. SGD meets this challenge by making random choices in some way, research this further.

## Updater RMSPROP

Without going into the updater in detail Note that momentum may be a hyperparameter that will need tuning on more complex networks. The problem in this demo is linear (? is it) but in a more complex graph with potential local minima momentum helps break through that. How deep do I go here?

## Layer 0 activation tanh

The activation function of a Layer determines the signal it sends to connected neurons.

Choices are sigmoid, smooth curve output 0-1 as x increases.

tanh similar to sigmoid output -1-> +1 depending on value of x

Stepwise output 0 or 1 depending on value of X

Etc going to deep here.

## Layer1 this is our output layer.

Note the activation is identity.

This determines that the output will be linear, a range of numeric values, .1, .2, .3 etc. VS 0 or 1, vs Class A, B or C

## STEP 3

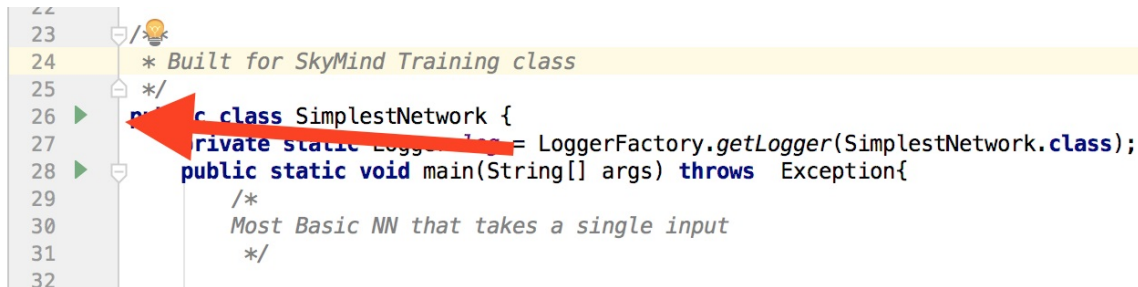
Run the code

In this step you will run the code.

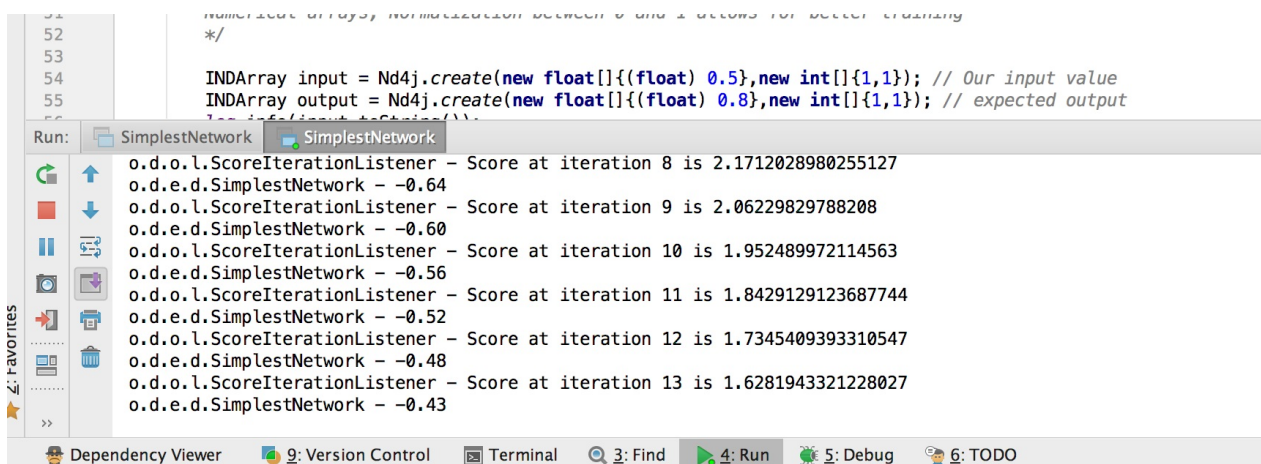
When the code executes it will create a UI that can be accessed with a web browser.

It will also print output to the output window at the bottom of IntelliJ as it runs

## Click on this green arrow to execute the code

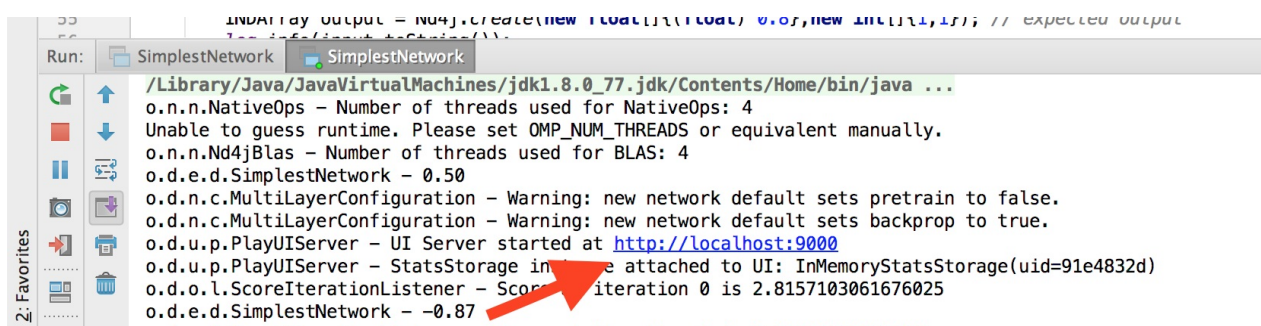


## View the output in the console while the class runs



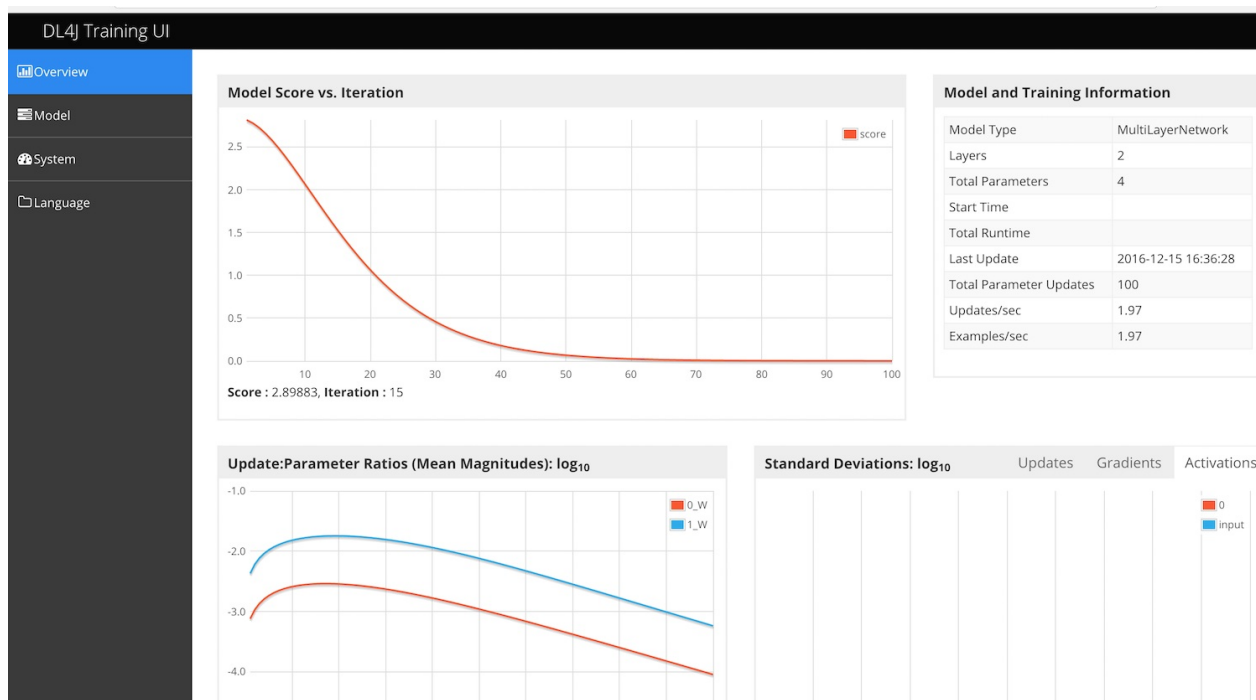
## View the UI

When the code executes and the UI is created, a line is generated in the console output with the url



Open that URL in a browser

You should see this



## Explanation of the output

Console Output.

The following block of code is what begins the training process.

```
for( int i=0; i<nEpochs; i++ ){
    model.fit(input,output);
    INDArray output2 = model.output(input);
    log.info(output2.toString());
    Thread.sleep(500);
}
```

## What is an Epoch?

It is a loop for the total number of Epochs. Or total passes through the training dataset, in this case our single input, but in real use cases it might be something like thousands of text reviews, or hundreds of thousands of images, or millions of lines from log files.

## What is Model.fit?

This is where the model trains. Data is ingested, random weights are assigned, output is evaluated against expected and weights are adjusted to lessen the error.

## What output should look like

This section

```
INDArray output2 = model.output(input);  
log.info(output2.toString());
```

Generates these lines in our console output.

```
o.d.e.d.SimplestNetwork - -0.87  
o.d.e.d.SimplestNetwork - -0.85
```

The "correct" output, or "expected" output is 0.80, you will see that the network is consistently getting closer to that goal as it trains.

This line in the console output

```
o.d.o.l.ScoreIterationListener - Score at iteration 1 is 2.775866985321045
```

Is generated by this line

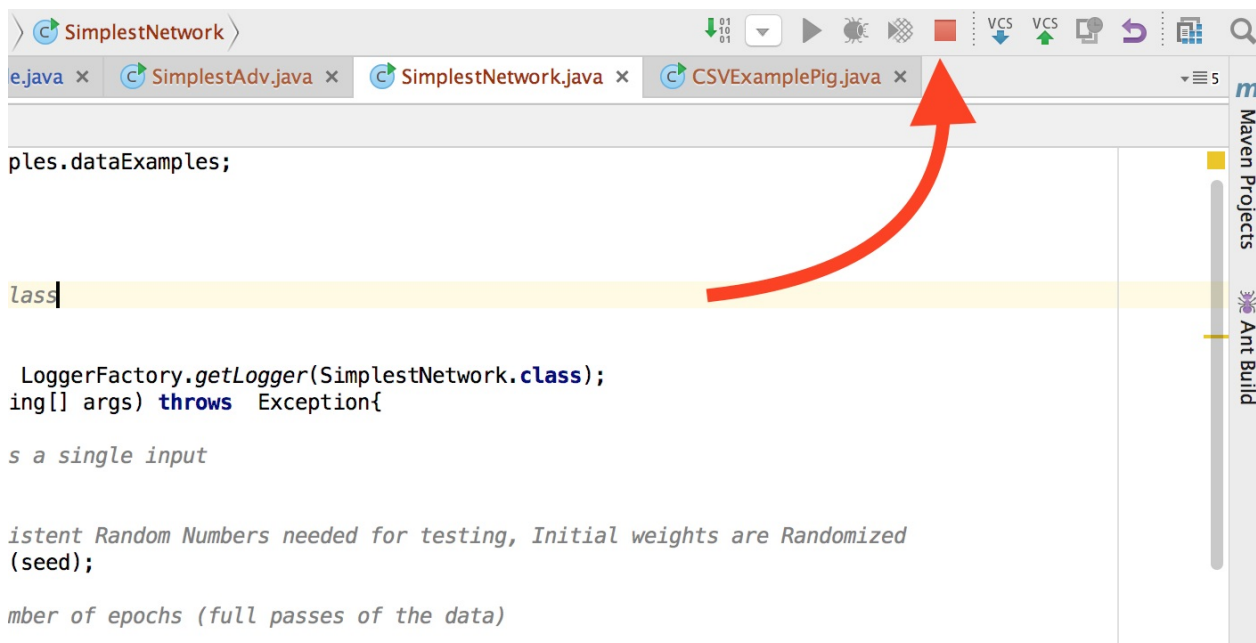
```
model.setListeners(new StatsListener(statsStorage), new ScoreIterationListener(1));
```

## STEP 4

In this step you will modify some of the parameters and see the effect on the training process.

Note that anytime you re-run this code you will have to terminate the previous running process. The webserver serving the UI will have a handle on a socket and the second example will try to grab that same socket, fail and return an error.

Kill the running process by clicking on the red square, top right.



Some parameters that you could tune.

Before you change things, note the current performance. How many iterations till it got to within .05 of the target? In 100 iterations how close did it get? Mine got to .78 after 100, and reached .75 at iteration 80

## Settings you may change with reasonable results

### Hidden Nodes

- Number of hidden nodes
  - Would provide more attempts towards the correct answer, more random weights, and may train quicker

### Number of Epochs

- Number of Epochs
  - If the network is converging on the target, then more epochs should allow it to get there, in time
  - Note that to prevent things going too fast to visualize, I put a sleep .5 seconds in the loop.
  - Remove that if you set to large number of Epochs.

## Learning Rate

Learning Rate Determines how far to adjust the weights given the error.

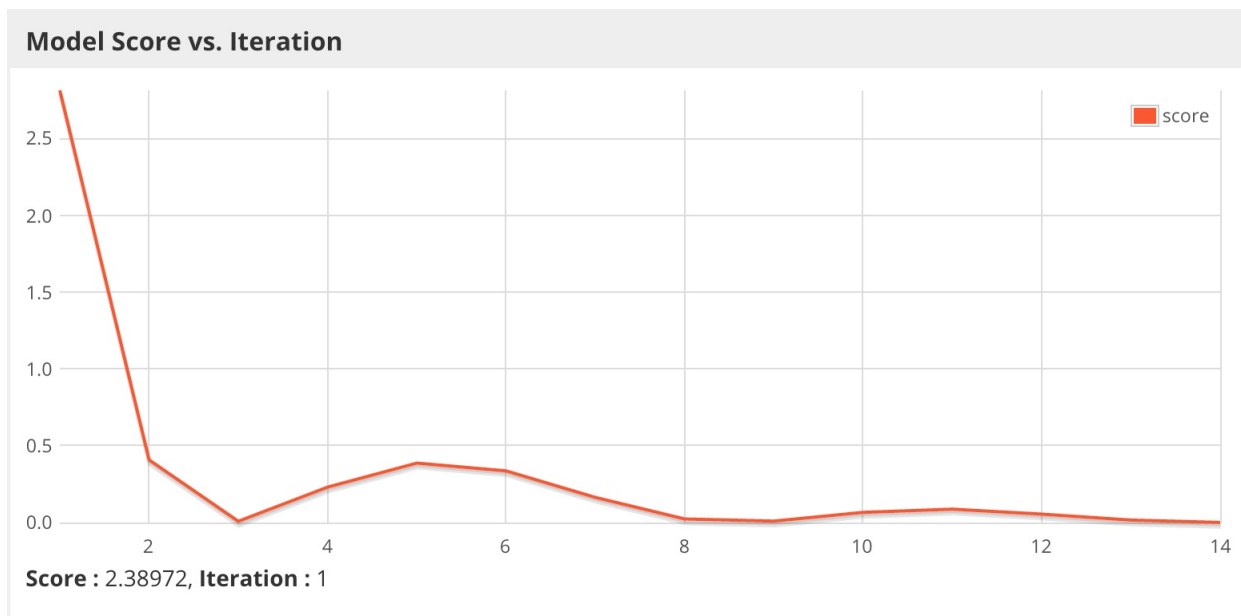
A range for learning rates would be ???

```
double learningRate = 0.001;
```

Change to perhaps...

```
double learningRate = 0.01;
```

Note that an aggressive learning rate may cause the network to overshoot the target before converging.



## Lab questions

1. What parameters may need adjusting in a Neural Net
- 2.

# Building a Convolutional Neural Network

A convolutional Neural Network differs from a Feed Forward Neural Network by having Convolutional Layers, followed by MaxPooling Layers.

A convolutional Layer looks at a subset of the image in a moving window across the complete image. As each region of the window is evaluated the model learns to detect useful features, regardless of where they occur in the image.

A MaxPooling Layer is used after a Convolutional layer. Pooling Layers function to reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.

---



## Lab Step 1:

Open the Class ConvNeuralNet and uncomment/delete the two lines with

`<-- DELETE THIS LINE --> .`

The open comment line

```
/* <-- DELETE THIS LINE -->
```

and the close comment line

```
<-- DELETE THIS LINE --> */
```

## Lab Step 2:

Most of the code here is exactly the same as the FeedForward Neural Network Example.

The Network will be modified and the two hidden layers will be replaced with a Convolutional Layer a MaxPooling Layer. The Output Layer will remain more or less the same.

Add the Convolutional Layer.

```
.layer(0, new ConvolutionLayer.Builder(5, 5)
    //nIn and nOut specify depth. nIn here is the nChannels and nOut is the number
    of filters to be applied
    .nIn(channels)
    .stride(1,1)
    .nOut(20)
    .activation(Activation.IDENTITY)
    .build())
```

## Lab Step 3:

Add the MaxPooling Layer.

```
.layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
    .kernelSize(2,2)
    .stride(2,2)
    .build())
```

## Lab Step 4:

Add the Output Layer

Note that this is very similar to the output Layer used in the FeedForward Network.

```
.layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD
)
    .nOut(outputNum)
    .activation(Activation.SOFTMAX)
    .build())
```

## Lab Step 5:

Run the Convolutional Neural Network and compare the accuracy to the accuracy of the FeedForward Network.

In a single epoch run the convnet may misclassify a 5 as a 6, or a 3 as an 8.

## Extra Credit:

Try to improve the performance of the Convolutional Network.

- Add more Layers

The Very Successful Image Recognition Neural Networks have 16 or more layers.

Add one or more pairs of Convolutional Layer followed by maxPooling Layers.

- Modify the size of the Convolution Layers

This line `ConvolutionLayer.Builder(5, 5)` Specifies a 5 pixel by 5 pixel window will be examined, and this line `.stride(1,1)` specifies to move the window over 1 pixel and repeat till end of line and then down one pixel.

Modifying those settings and see if you can get better results.

# Saving and Loading a Trained Model

In this lab you will

1. Attach a UI instance to a model
2. Save the model
3. Load the saved model

## Goals of the Lab

- Label Generation with ParentPathLabelGenerator
- Attaching a UIServer
- Saving a Trained model
- Loading a Trained model

## Step 1

- Open up IntelliJ Open up IntelliJ and navigate to the Labs folder

## Step 2

- Open the MnistImageSave class

This example uses the classic Machine Learning challenge of recognizing handwritten digits.

## Step 3

- Review the Input

The data is in the following directories `resources/mnist_png/testing`, `resources/mnist_png/training`.

Open one of the directories and note that it contains subdirectories 0-9 each containing 24\*24 pixel gray scale images of digits.

## STEP 4

- Attach a UI instance to the Neural Network.

See the SimplestNetwork for an example.

## Step 5

- Save the Model

The Javadoc for ModelSerializer is here

<https://deeplearning4j.org/doc/org/deeplearning4j/util/ModelSerializer.html>

You want to use ModelSerializer.writeModel with the constructor that takes(modelname, location\_to\_save, boolean saveUpdater)

The modelname will be the name of the model in this example.

The location to save is already defined. File locationToSave = new File("trained\_mnist\_model.zip");

SaveUpdater is used when you want to train the model further or simply use it for inference.

***some models train for days or weeks, with any longrunning process it is important to save progress in case of disaster. Running ModelSerializer.writeModel allows you to save the model, perhaps every few hours.***

## Step 6

Run the code

## Step 7

Load the Trained model.

Create a class named MnistImageLoad

add a Logger

```
private static Logger log = LoggerFactory.getLogger(MnistImageLoad.class);
```

Add a Main method

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

Set three Int objects height, width, channels.

Set values to 28,28 and 1.

Move the saved model to the resources folder so it is in the classpath.

Create two file objects one for the trained\_mnist\_model.zip file and one for the image that will be classified by the model, number.png.

```
File modelSave = new ClassPathResource("trained_mnist_model.zip").getFile();
```

```
File imageToTest = new ClassPathResource("number.png").getFile();
```

Create a MultiLayerNetwork model using ModelSerializer.restoreMultiLaerNetwork([File Object for the model]);

To read a single image to pass into the saved network for evaluation use the NativeImageLoader Class.

```
NativeImageLoader loader = new NativeImageLoader(height, width, channels);
```

Get the image into an INDarray

```
INDArray image = loader.asMatrix(imageToTest);
```

For expected results the image must be processed in the same way as the images that the network trained on were processed.

Create a DataNormalization scaler to scale pixel values to values between 0 and 1.

```
DataNormalization scaler = new ImagePreProcessingScaler(0,1);
```

Transform the image with the transform method of the DataNormalization scaler.

```
scaler.transform(image);
```

Pass the transformed image matrix to neural Net.

```
INDArray output = model.output(image);
```

Print the output to the console.

```
log.info(output.toString());
```

Run your code to verify.