# Table of Contents


Introduction 1.1

Simplest Network Lab 1.2

- Simplest Network Lab 1.2.1

DataVec Lab 1.3

- DataVec Lab 1.3.1

LSTM Character Generation 1.4

- LSTM Lab 1.4.1

# Introduction to DeepLearning4J LAB GUIDE

# Simplest Network Lab

# 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 Algoithm 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

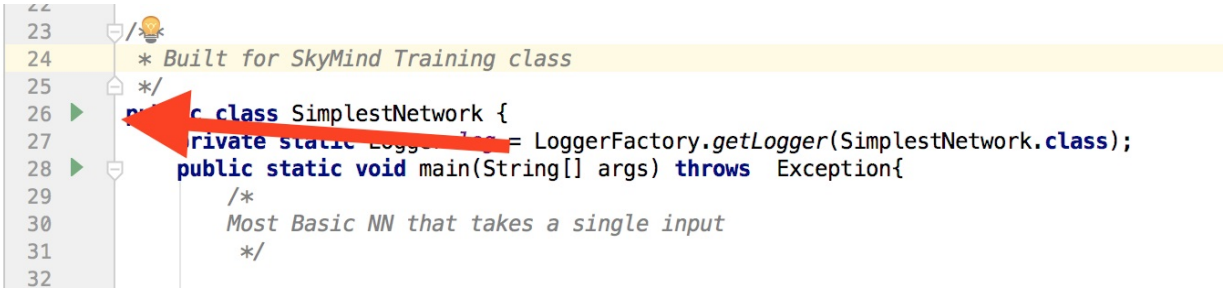
Run the code

In this step you will run the code.

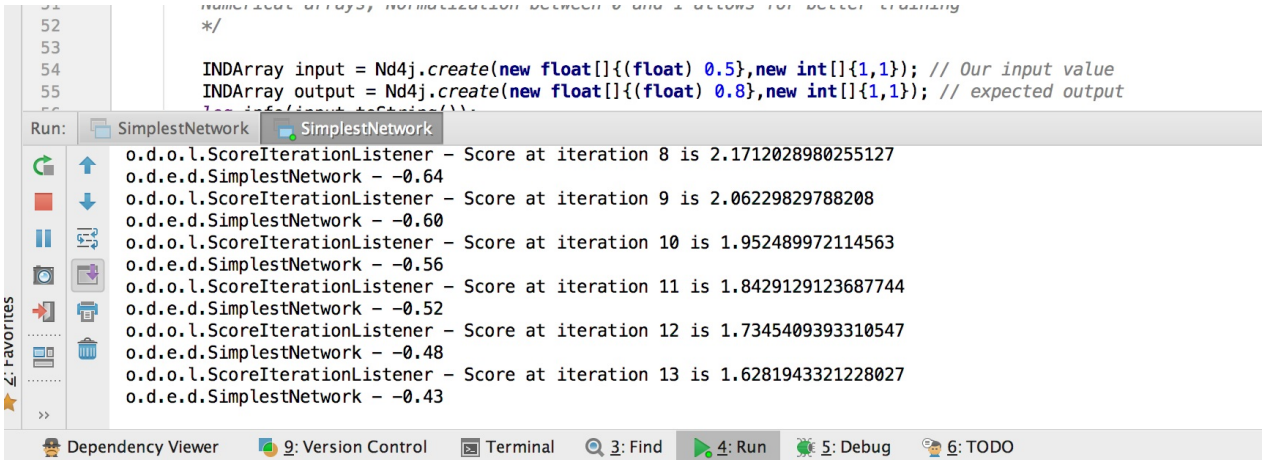
When the code executes it will create a UI that can be accessed with a web browser.

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

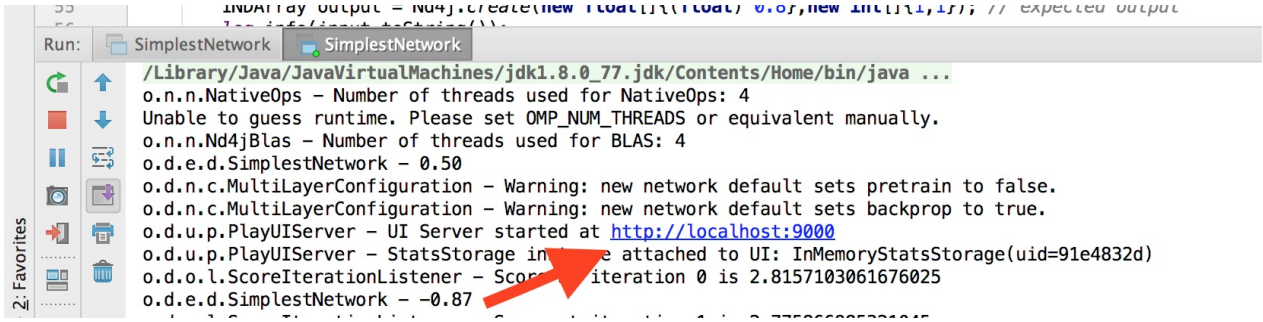



# View the output in the console while the class runs



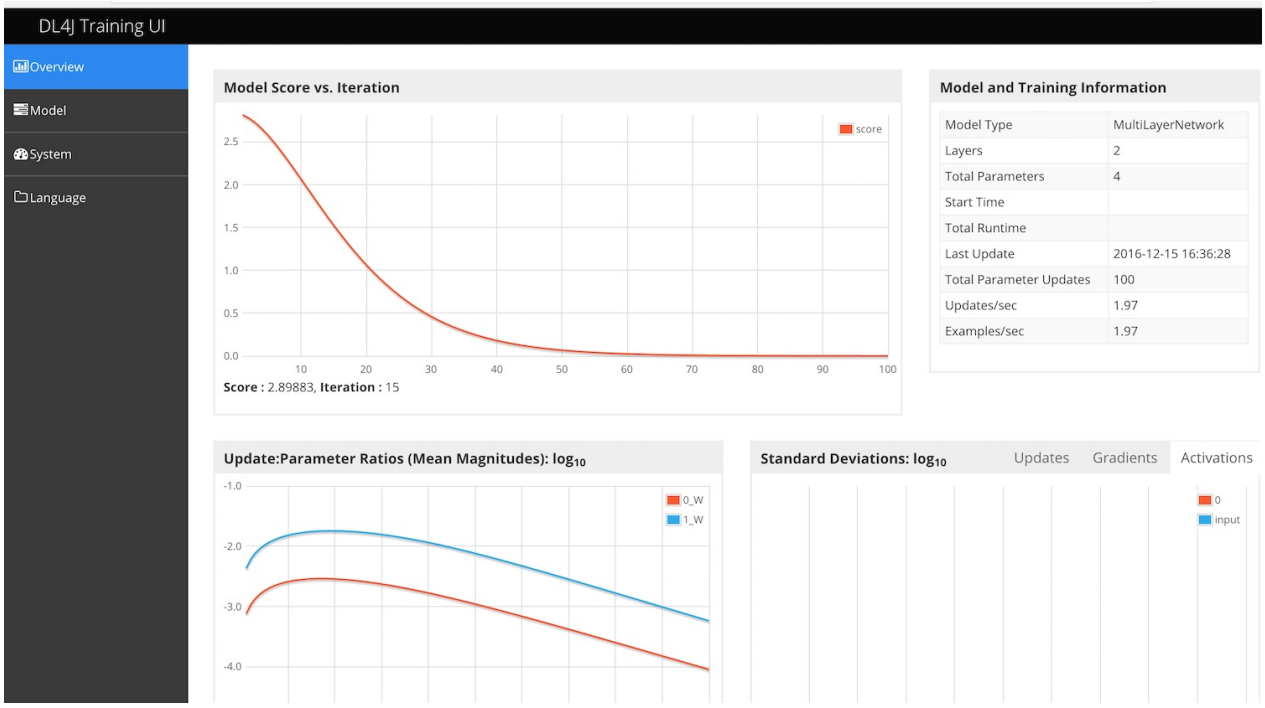

# View the UI

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




# Explaination 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 milllions od 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.

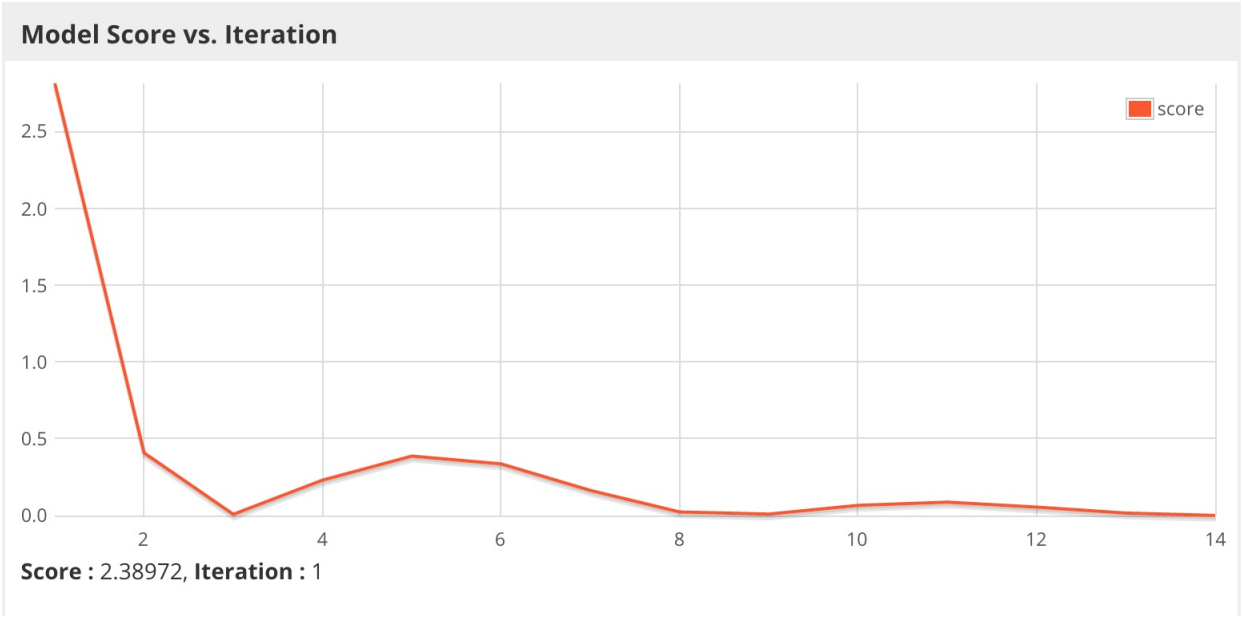
A range for learning rates would be ???

```
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
2.

# 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 Iris DataVec class

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

## Step 3

- Review the Java Code

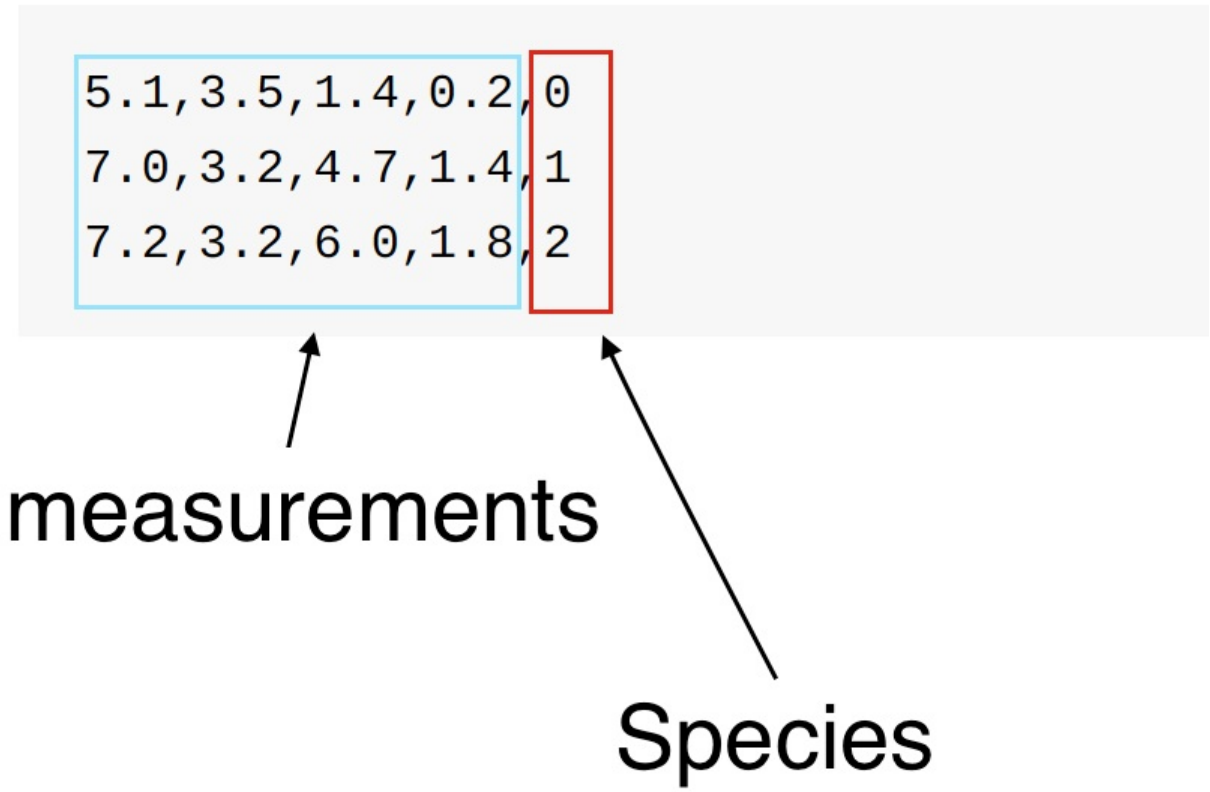
In this case the Neural Network has already been built. The goal of this Lab is to work through the Data ETL process using DataVec.

- 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

```
5.1,3.5,1.4,0.2,0
7.0,3.2,4.7,1.4,1
7.2,3.2,6.0,1.8,2
```




# STEP 3

Review the needed steps.

1. Read the File
2. Parse the Lines
3. Specify Label fields vs measurment

4. Create a DataSet object to pass into our Nerual 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 delimeted.

** 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 verication scripts to verify every line has the same amount of commas, at the very least.

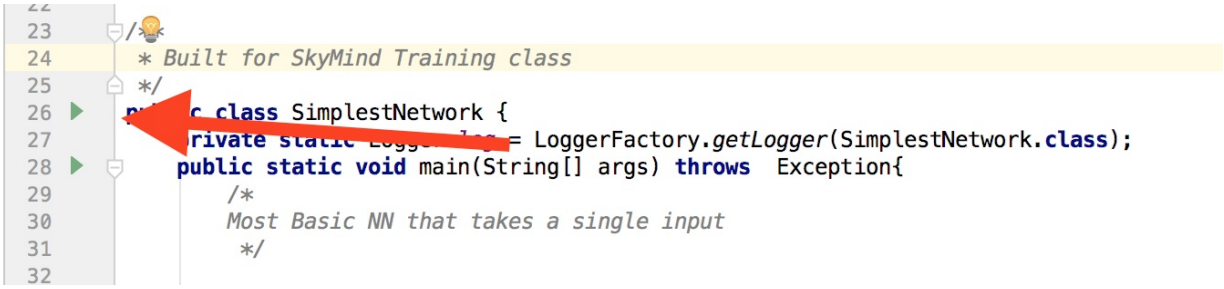
After verifying that there are no header lines, and the delimiter is a comma, add the following code to the stub program.

```
    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.

`new ClassPathResource(fileName).getFile()` access a file on the ClassPath

A record Reader is initialized and passed 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();

# Create a DataSet Iterator

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

# Shuffle the Data

allData.shuffle();

# Split Train and Test

```
    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

# Extra Credit

Parent Path Label Generator

# Simplest Network 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 begining of line denotes begining of phrase, no "." at begining 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 begining 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 WeatherForecast Class

Click on WeatherForecast.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 Optimization Algoithm 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

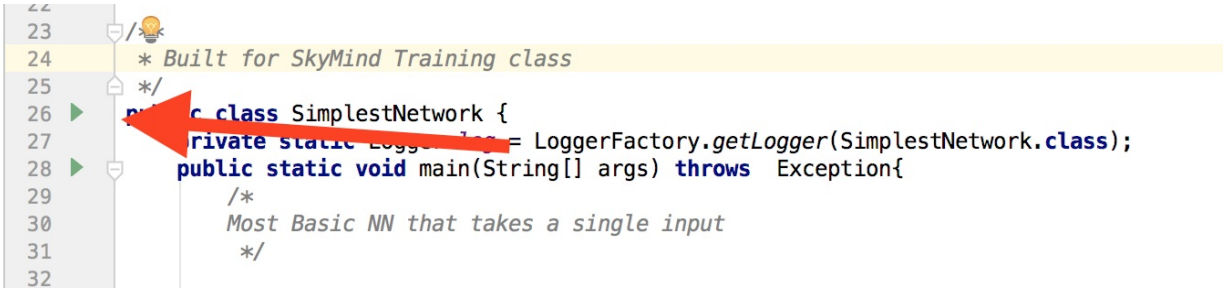
Run the code

In this step you will run the code.

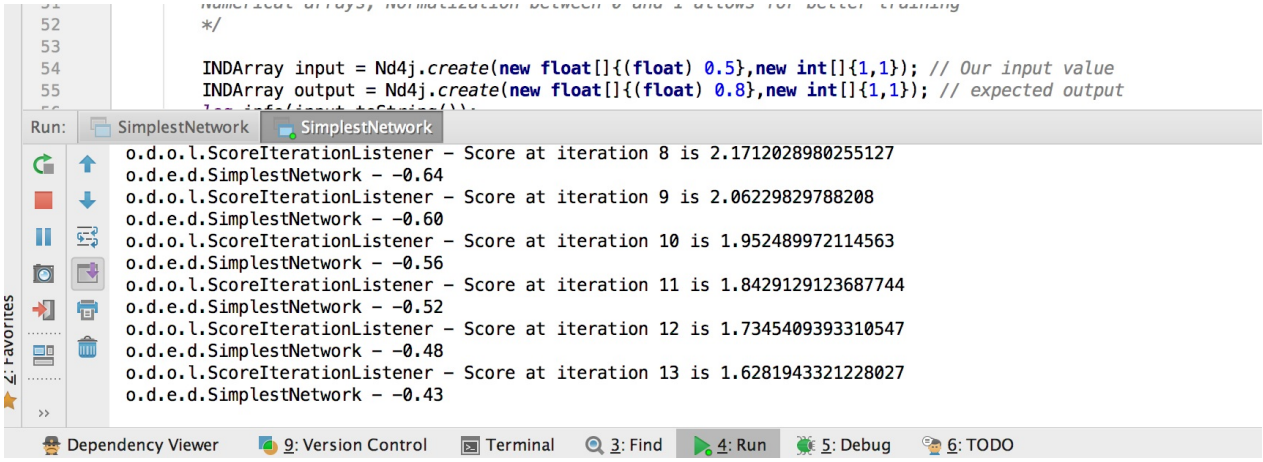
When the code executes it will create a UI that can be accessed with a web browser.

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

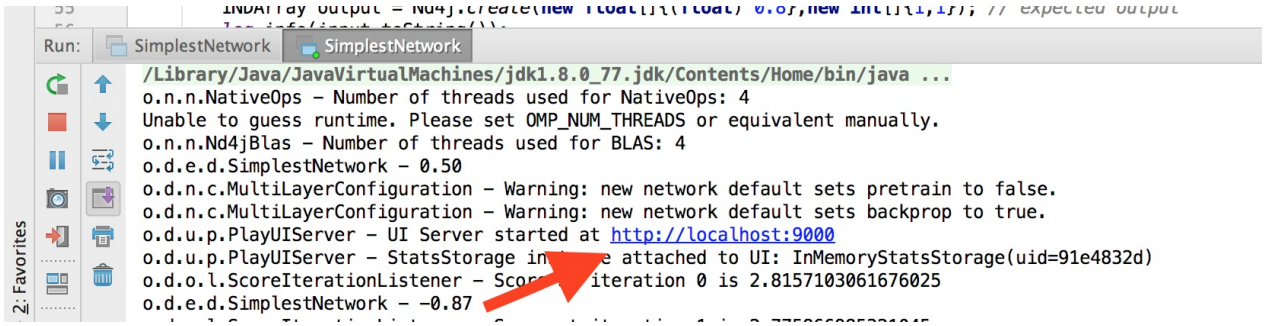



# View the output in the console while the class runs



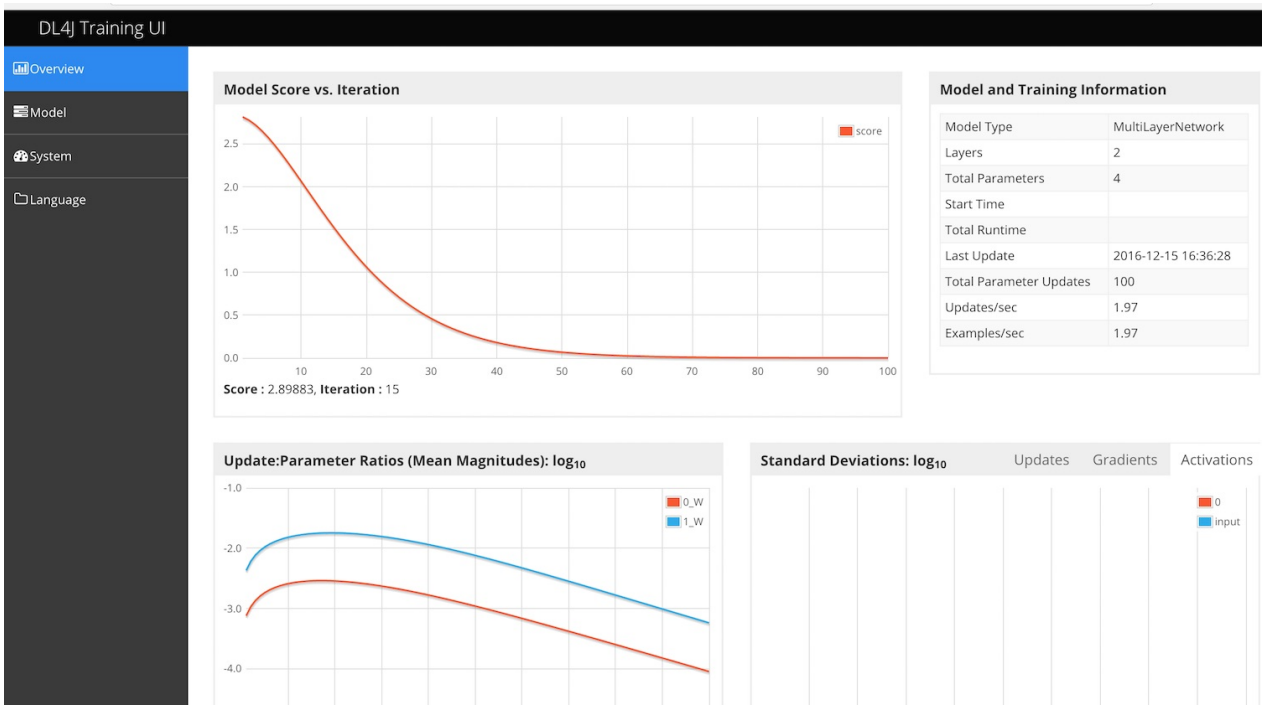

# View the UI

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




# Explaination 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 milllions od 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.

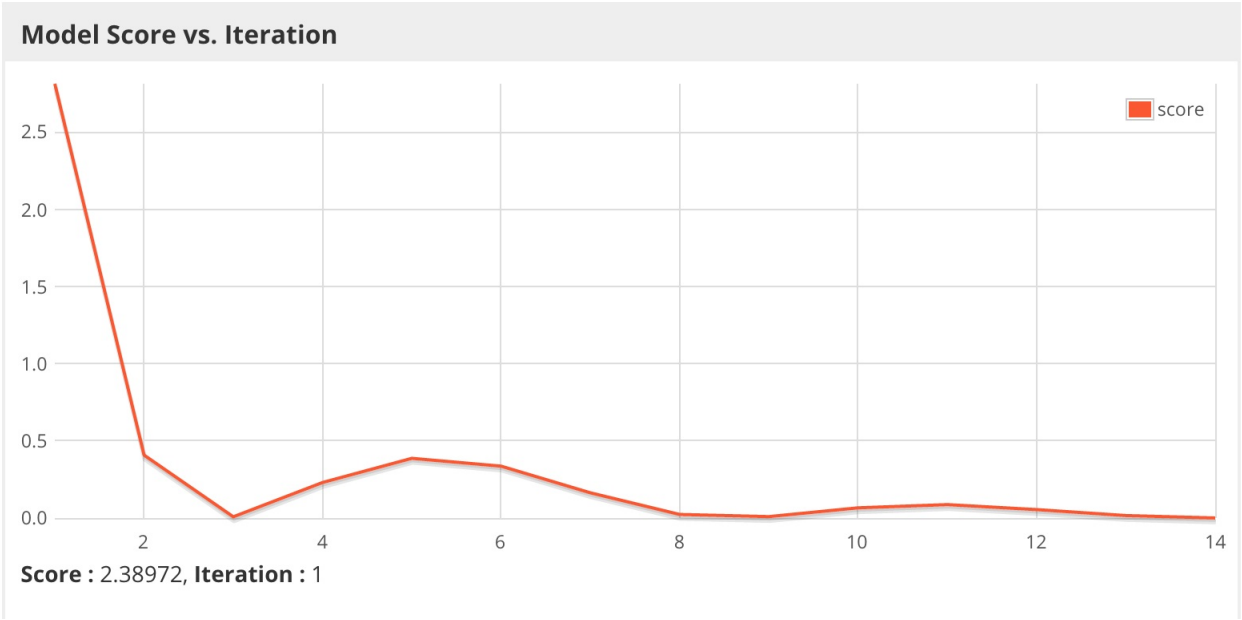
A range for learning rates would be ???

```
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

2.