Table of Contents

Skymind 2017 Strata Tutorial

Josh Patterson, Dave Kale, Susan Eraly

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

Setting up your Virtual Enviroment

1. Save the Strata_2017_Labs.ova file
2. Import the Appliance into VirtualBox, VMware should work as well

Logging into your VM

- Username Pass

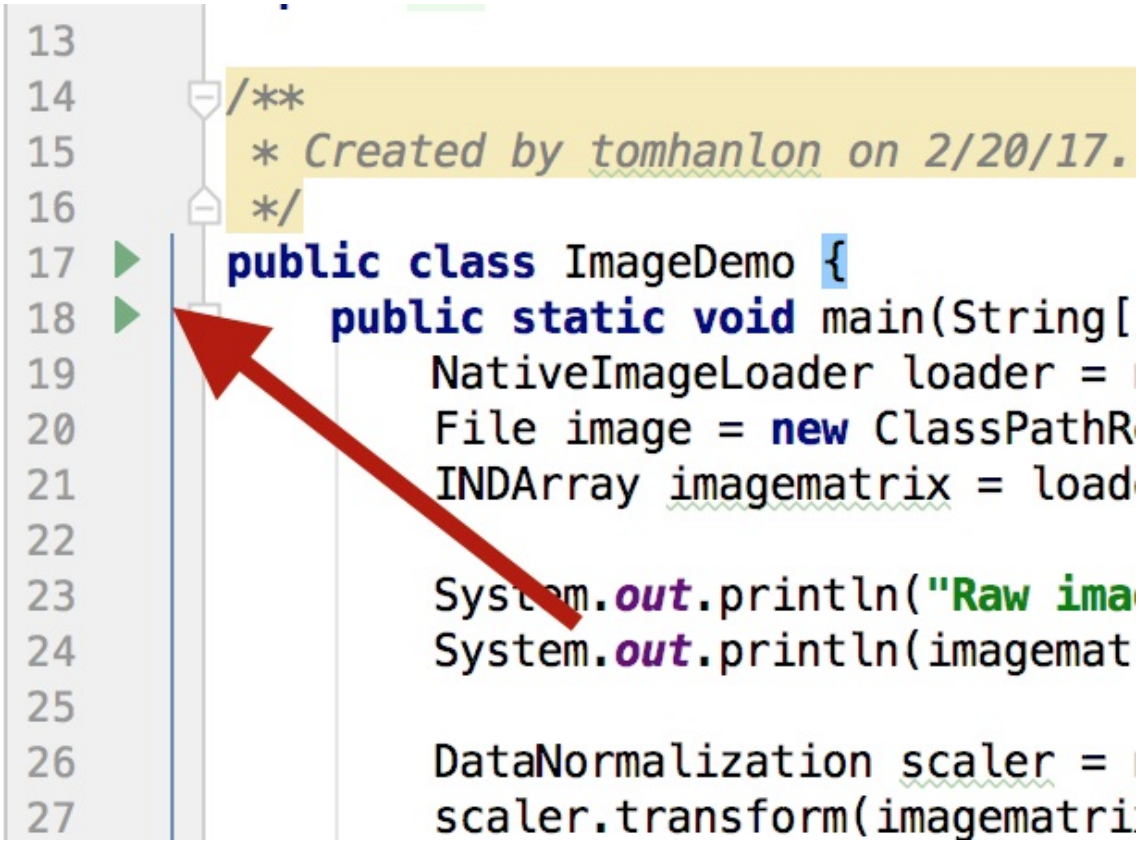
skymind:skymind

- SSH ssh -p 2222 skymind@localhost
- Copying files with SCP scp -P 2222 skymind@localhost:/path/to/source /path/to/destination

Running and stopping code

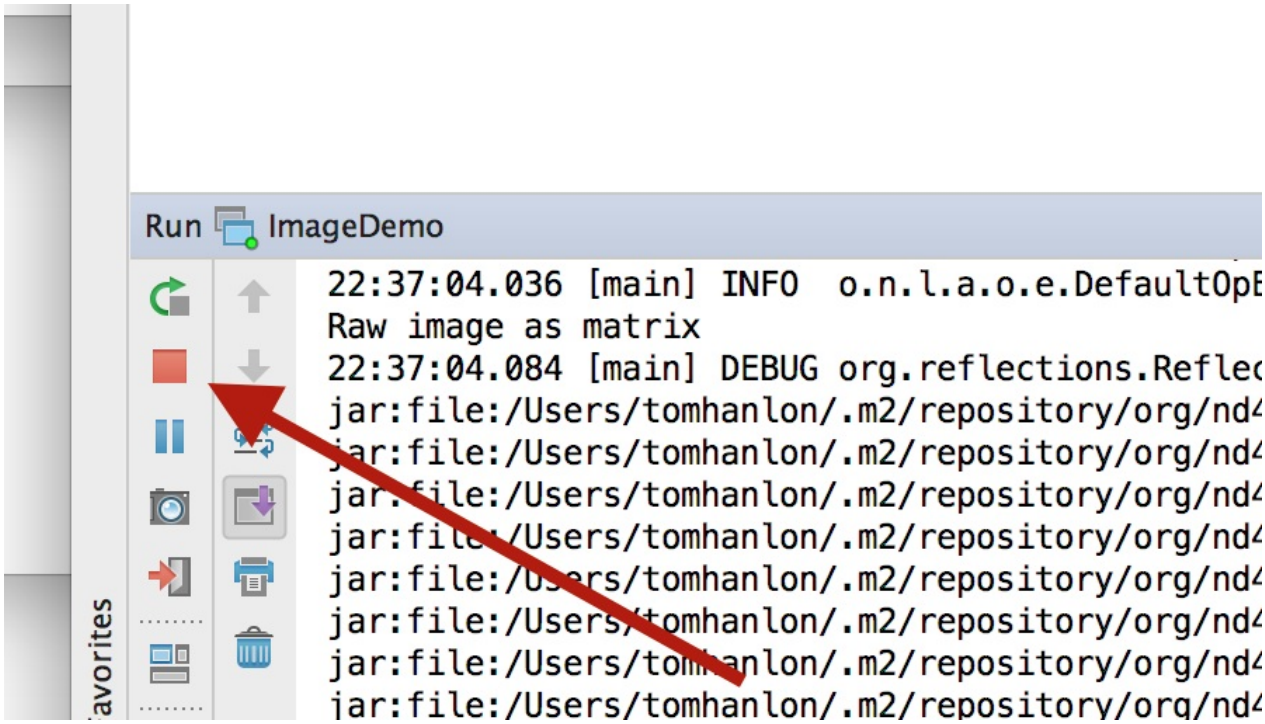
When editing or creating a java class you can run it at any time by hitting the green arrow next to your main class.

- Run your code



- Stop your code

To stop the code you can hit the red button.



1. Common issues

- Class won't run due to webui conflict

A lot of the classes launch a webUI, that WebUi will continue to run and continue to own port 9000 locally. If you run another class that has a webui it will try to bind to port 9000 and fail, returning an error.

Solution: Kill all other running classes and try again.

- Class won't run due to another class in the project failing to compile

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

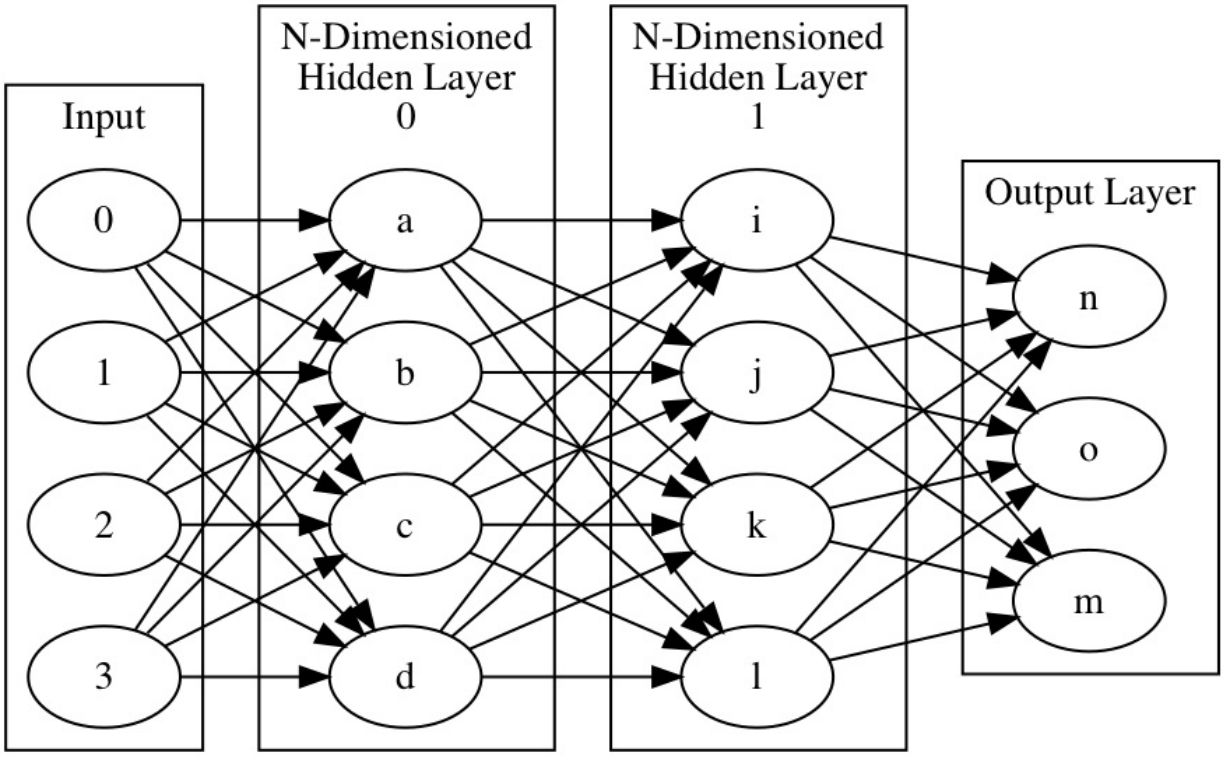
You are welcome to start a blank project and work from scratch. If your questions are related to the course materials or the topic the instructor will happily answer your questions

Building a Feed Forward Neural Net With DeepLearning4J

What is a Feed Forward Neural Network

- A Neural network with X numbers of fully connected Layers

Feed Forward Neural Net:



Lab Introduction

- Feed Forward Neural Network with 2 hidden layers
- Learning to Recognize and Classify Images
  - 28 * 28 Grayscale

Image examples

Exploring the dataset

- Data stored in

```
resources/mnist_png
```

- Pre-split into training and testing
- stored in labelled directories

Lab Step 1:

Open Intellij and navigate to the Labs project.

The project contains a Labs directory with stub programs for you to complete, a demos directory with demos, and a solutions directory with solutions for the lab.

```
src/main/java/ai.skymind.training/labs
```

Lab Step 2:

Set some image related variables for later use.

The Mnist images are 28 * 28 grayscale.

```
int height = 28;
int width = 28;
int channels = 1;
```

Add three lines to the code stub specifying height width and channels.

Lab Step 3:

Set an int for Random Number Seed. Same seed with different runs allow for consistent results.

```
int rngseed = 123;
Random randNumGen = new Random(rngseed);
```

Intellij will highlight the code in red, until the needed import statement is added "import java.util.Random;"

Lab Step 4:

Set Batchsize. Batchsize determines how many records to train on before adjusting weights. In this case we are classifying to 10 possible classes, our batchsize should be large enough and our data randomized to include records from each class in the batch.

Add a line to set the batchsize.

```
int batchSize = 128;
```

Lab Step 5:

For Classification the number of nodes in the output layer is equal to the number of classes.

Add this line to the code

```
int outputNum = 10;
```

Lab Step 6:

An epoch is a total pass through the training data. Set numEpochs to 15

Add this line to the code

```
 int numEpochs = 15;
```

Lab Step 7:

Add these two lines to define the paths to the train folder and the test folder

```
File trainData = new ClassPathResource("mnist_png/training").getFile();
File testData = new ClassPathResource("mnist_png/testing").getFile();
```

The needed imports for this section are.

```
import java.io.File;
import java.util.Random;
```

Lab Step 8:

Define FileSplit for test and train by providing from File Path,Allowed Formats, and Random Number to FileSplit

```
FileSplit train = new FileSplit(trainData, NativeImageLoader.ALLOWED_FORMATS,randN
umGen);
        FileSplit test = new FileSplit(testData,NativeImageLoader.ALLOWED_FORMATS,
randNumGen);
```

The needed imports for this step are:

```
import org.datavec.api.split.FileSplit;
import org.datavec.api.util.ClassPathResource;
```

Lab Step 9:

Create a Parent Path Label Generator to take the directory name as the label for the image.

```
ParentPathLabelGenerator labelMaker = new ParentPathLabelGenerator();
```

The import required for this step is:

```
import org.datavec.api.io.labels.ParentPathLabelGenerator;
```

Lab Step 10:

We will need two RecordReaders, one for the training data and one for the test data.

Create the RecordReaders and proved height, width, channels, and LabelMaker.

```
ImageRecordReader recordReader = new ImageRecordReader(height,width,channels,label
Maker);
ImageRecordReader recordReaderTest = new ImageRecordReader(height,width,channels,l
abelMaker);
```

The import required for this step is:

```
import org.datavec.image.recordreader.ImageRecordReader;
```

Lab Step 11:

Initialize the recordreaders.

```
recordReader.initialize(train);
recordReaderTest.initialize(test);
```

Lab Step 12:

A DataSetIterator builds an INDarray from the List of Writables that the RecordReader provides.

Create two iterators one for the test data and one for training data.

```
DataSetIterator dataIter = new RecordReaderDataSetIterator(recordReader,batchSize,
1,outputNum);
DataSetIterator testIter = new RecordReaderDataSetIterator(recordReaderTest,batchS
ize,1,outputNum);
```

The imports required for this step are:

```
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
```

Lab Step 13:

Scale the pixel values between 0 and 1 with an ImagePreProcessingScaler.

Scaling the data will give better results.

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

The imports required for this step are:

```
import org.nd4j.linalg.dataset.api.preprocessor.DataNormalization;
import org.nd4j.linalg.dataset.api.preprocessor.ImagePreProcessingScaler;
```

Lab Step 14:

Call the fit method on the scaler.

```
scaler.fit(dataIter);
```

Lab Step 15:

Apply the scaler to the training iterator and the testing iterator using the setPreProcessor method of the datasetiterator.

```
dataIter.setPreProcessor(scaler);
testIter.setPreProcessor(scaler);
```

Lab Step 16:

Build the Neural Network.

The code for the Neural Network is available as a stub. Uncomment and examine the code.

The imports required for this section ***steps 16,17,18,19*** are:

```
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.Updater;
import org.deeplearning4j.nn.conf.inputs.InputType;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.weights.WeightInit;
import org.nd4j.linalg.activations.Activation;
```

This first section defines parameters applicable to all layers of the network.

The Stub is complete except for 3 important settings, in this step you will set the

- OptimizationAlgorithm
- learningRate
- Updater

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(rngseed)
            .optimizationAlgo(OptimizationAlgorithm.### YOUR CODE HERE ###)
            .iterations(1)
            .learningRate(###YOUR_CODE_HERE)
            .updater(Updater.###YOUR CODE HERE###).momentum(0.9)
            .regularization(true).l2(1e-4)
            .list()
```

Set the OptimizationAlgorithm to STOCHASTIC_GRADIENT_DESCENT

Set the learning Rate to 0.006

Set the Updater to Nesterovs.

Lab Step 17:

The next section of the code uncommented in the above step defines a hidden layer. You need to add three parts, defining the nIn, the Activation, and the Initial Weights.

```
.layer(0, new DenseLayer.Builder()
    .nIn(###YOUR CODE HERE####)
    .nOut(100)
    .activation(Activation.#### YOUR CODE HERE ####)
    .weightInit(WeightInit.#### YOUR CODE HERE ####)
    .build())
```

Set the nIn to "height * width", the height and width of the input images.

Set the Activation for this layer to "RELU".

Set the WeightInit to "XAVIER".

Lab Step 18:

Complete the code for the output layer.

```
.layer(1, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD
)
    .nIn(### YOUR CODE HERE ###)
    .nOut(### YOUR CODE HERE ###)
    .activation(Activation.### YOUR CODE HERE ###)
    .weightInit(WeightInit.### YOUR CODE HERE ###)
    .build())
```

Set the nIn for this layer to match the nOut for the previous layer.

This Network is performing classification, set the nOut to equal the number of classes. In this case the number of digits, 0-9, "10".

To convert the output of each output node to a probability per class, set the activation to "SOFTMAX"

Set the weight initialization algorithm to "XAVIER"

Lab Step 19:

Complete the final block of code defining the configuration of the Neural Network

```
.pretrain(### YOUR CODE HERE ###).backprop(### YOUR CODE HERE ###)
    .setInputType(InputType.convolutional(height,width,channels))
    .build();
```

Pretrain applies to specialized neural networks that do unsupervised learning, it does not apply to this simple feed forward network.

Set pretrain to "false"

Backprop is how the neural network trains, it should be set to true.

Set backprop to "true"

Lab Step 20:

Uncomment the rest of the code by removing the lines with `<-- remove this line -->`.

You will need the following imports:

```
import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.deeplearning4j.ui.api.UIServer;
import org.deeplearning4j.ui.stats.StatsListener;
import org.deeplearning4j.ui.storage.InMemoryStatsStorage;
```

Lab Step 21:

View the Web Based User Interface as the network trains by opening a browser

http://localhost:9000/

Keras Model Import Lab

Introduction

- Keras
  - Popular Python tool for Deep Learning
- DeepLearning4J
  - Production worthy Java based Deep Learning
- Keras Model Import
  - Best of both worlds

A Keras Model

- resources/Keras/iris.py

```
import numpy
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.cross_validation import cross_val_score, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
print(X)
print(Y)

#encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)


# convert integers to dummy variables (hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
print(dummy_y)

# define baseline model
#def baseline_model():
# create model
model = Sequential()
model.add(Dense(4, input_dim=4, activation='relu'))
model.add(Dense(3,activation='sigmoid'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
```

```
#    return model
#model.fit
model.fit(X, dummy_y, nb_epoch=200, batch_size=5)
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]));
print(prediction);



# To casve just the weights
model.save_weights('/tmp/iris_model_weights')

# To save the weights and the config
# Note this is what is used for this demo
model.save('/tmp/full_iris_model')

# To save the Json config to a file
json_string = model.to_json()
text_file = open("/tmp/iris_model_json", "w")
text_file.write(json_string)
text_file.close()
```

Iris DataSet

- Flower data
  - resources/Keras/iris.csv

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
```

Saving a Keras Model

```
model.save('/tmp/full_iris_model')
``



------------------
<div style="page-break-after: always;"></div>

# Lab Step 1

In this lab we have run the python Keras code and the saved model and weights are stored in ``` resources/Keras/full_iris_model
```

Create a String for the filepath to the full_iris_model file.

We use DataVec's ClassPathResource here to get information on files in our Classpath by providing a name of the file. The String for the classpath could be provided directly, we use ClassPathResource so we can drop a file in the resources directory and have a portable demonstration environment.

```
String kerasModelfromKerasExport = new ClassPathResource("Keras/full_iris_model").
getFile().getPath();
```

Lab Step 2

Create a MultiLayerNetwork by using `KerasModelImport.importKerasSequentialModelAndWeights` and providing a String for the FilePath to the saved Keras Model.

```
 MultiLayerNetwork model = KerasModelImport.importKerasSequentialModelAndWeights(k
erasModelfromKerasExport);
```

The required import for this step is:

```
import org.deeplearning4j.nn.modelimport.keras.KerasModelImport;
```

Lab Step 3

Add some code to verify the model import was successfull. In this step you create the Input Array to test with.

The Keras model was trained and then tested with a single input.

```
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]))
```

The output showed high probability for Class 1, less for class 2, and even less for class 3.

```
[[ 0.92084521  0.13397516  0.03294737]]
```

Add this code to the class.

```
INDArray myArray = Nd4j.zeros(1, 4); // one row 4 column array
        myArray.putScalar(0,0, 4.6); // first value sepal length
        myArray.putScalar(0,1, 3.6); // second value sepal width
        myArray.putScalar(0,2, 1.0); // third value petal length
        myArray.putScalar(0,3, 0.2); // fourth value petal width
```

Lab Step 4

Validate the model is working and results match the Keras model.

Add this to the class.

```
INDArray output = model.output(myArray);
        System.out.println("First Model Output");
        System.out.println(myArray);
        System.out.println(output);
```

Lab Step 5

Run your code and verify the output.

You should see on the console output the following values.

```
[4.60, 3.60, 1.00, 0.20]
[0.92, 0.13, 0.03]
```

The first line is our input INDArray The second line is our output.

Does it match the prediction of the model when run in Keras?

Congratulations you have imported a Keras Model into DeepLearning4J.

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 hiddel 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 Succesful 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.

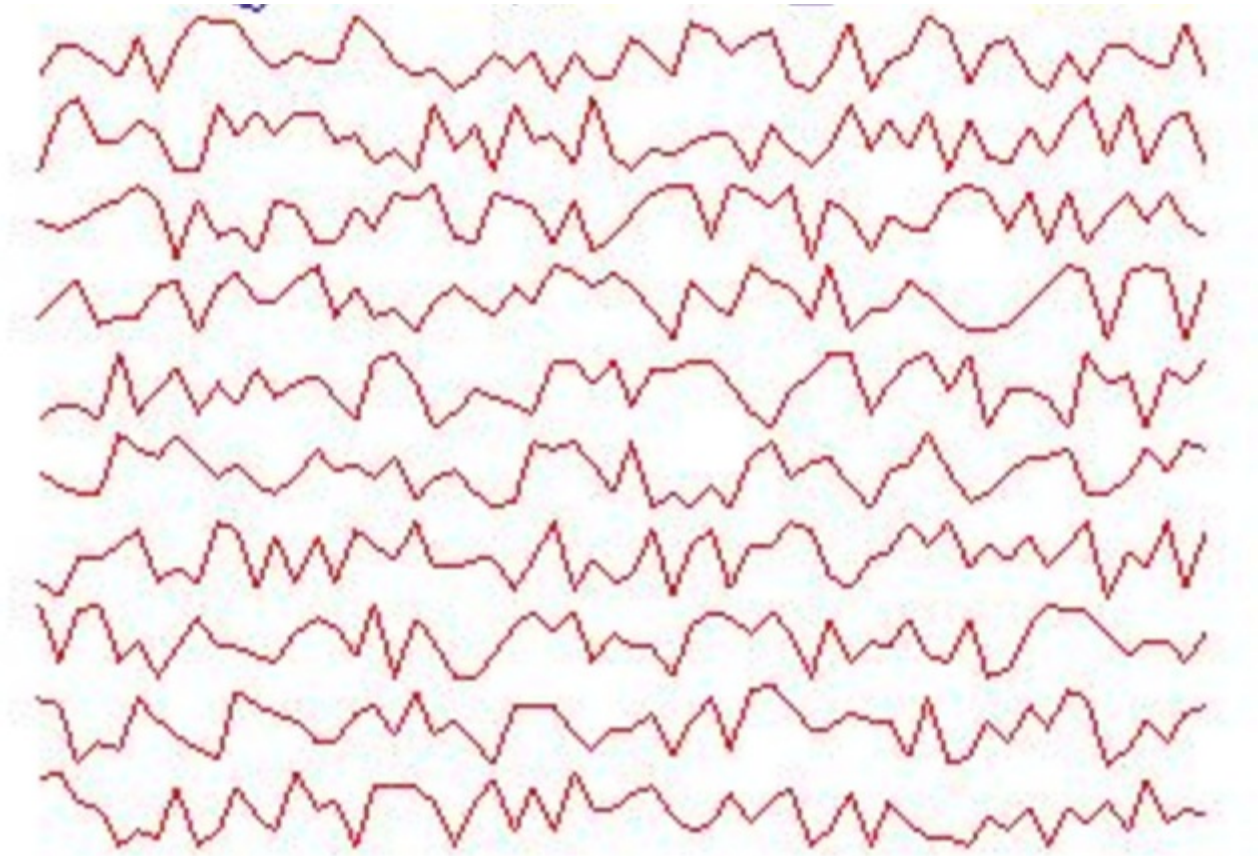
Modifying those settings and see if you can get better results.

Sequence Classification Lab

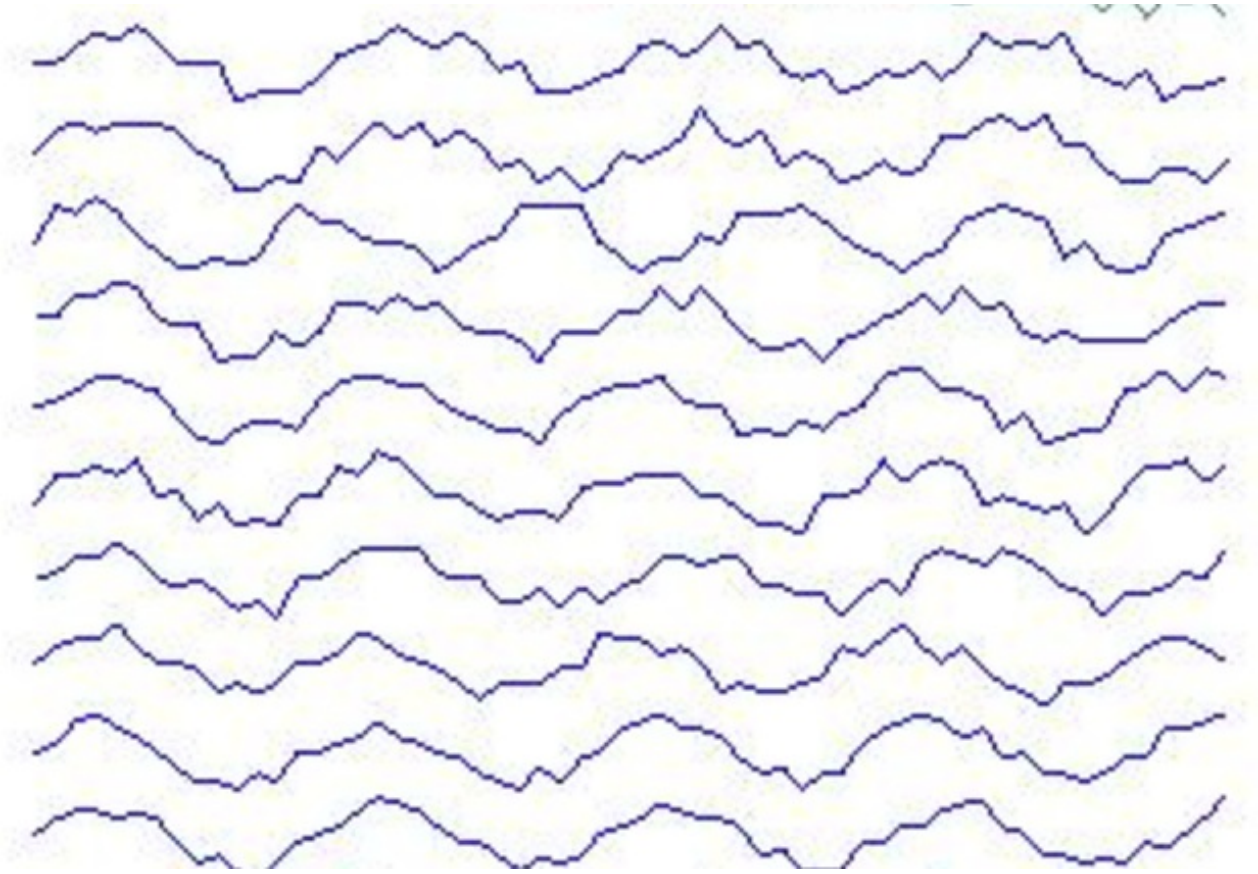
This Lab uses an LSTM Recurrent Neural Network to classify Sequence Data into 6 different classes.

The classes are:

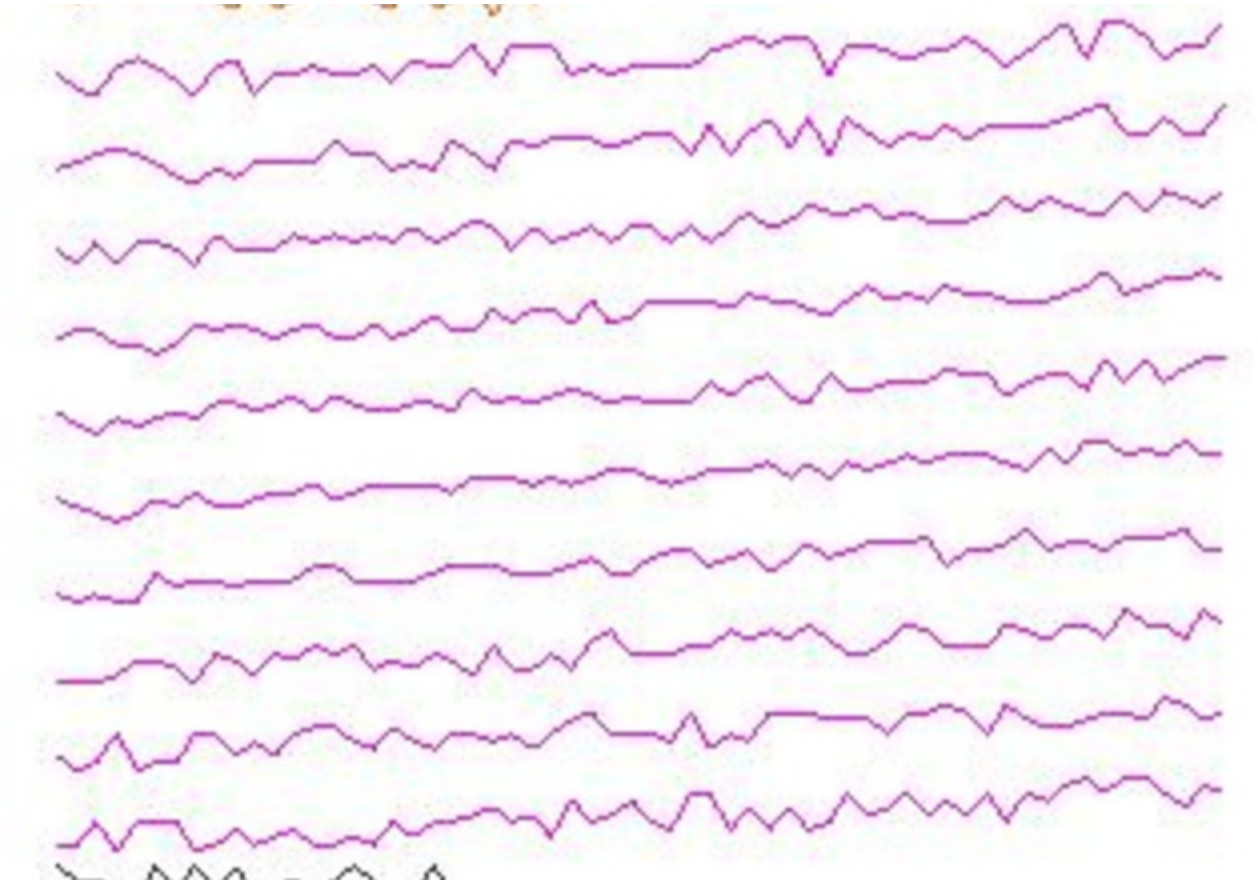
Normal



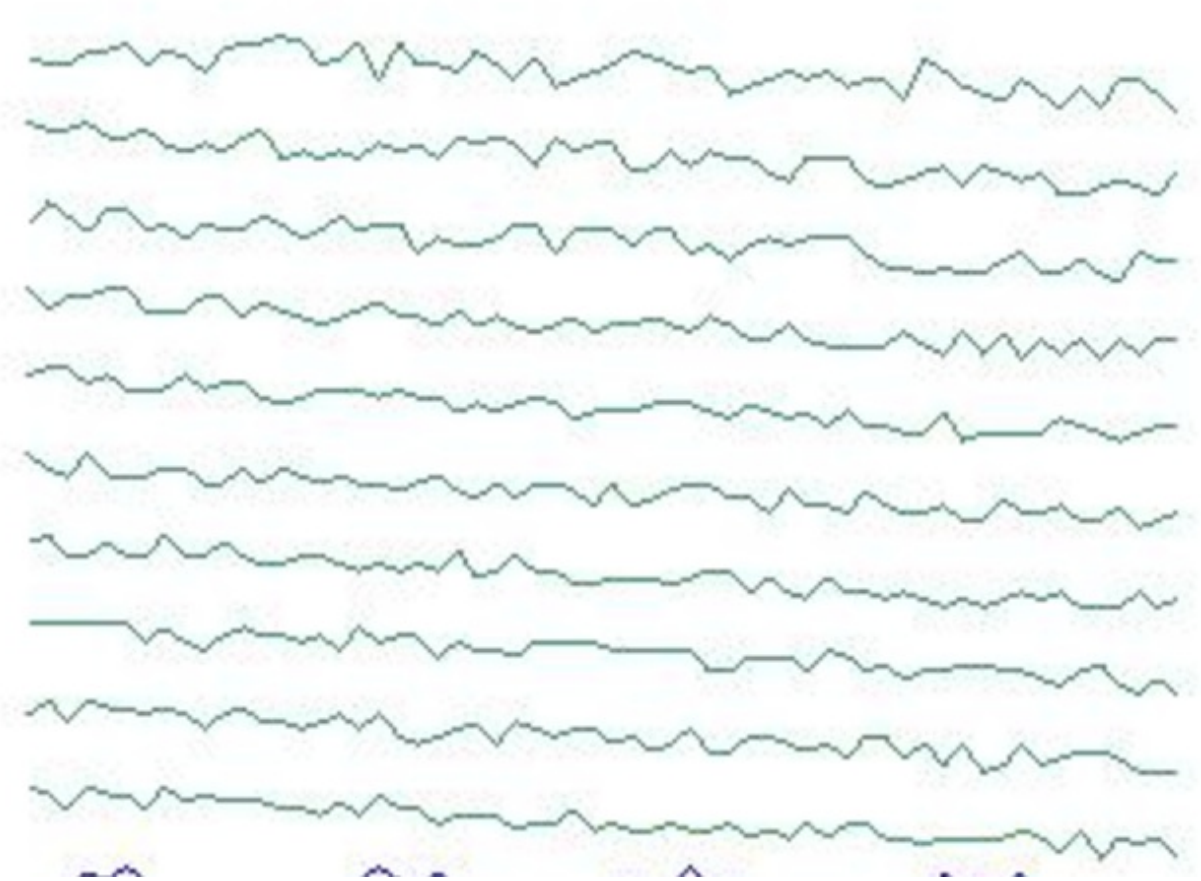
Cyclic



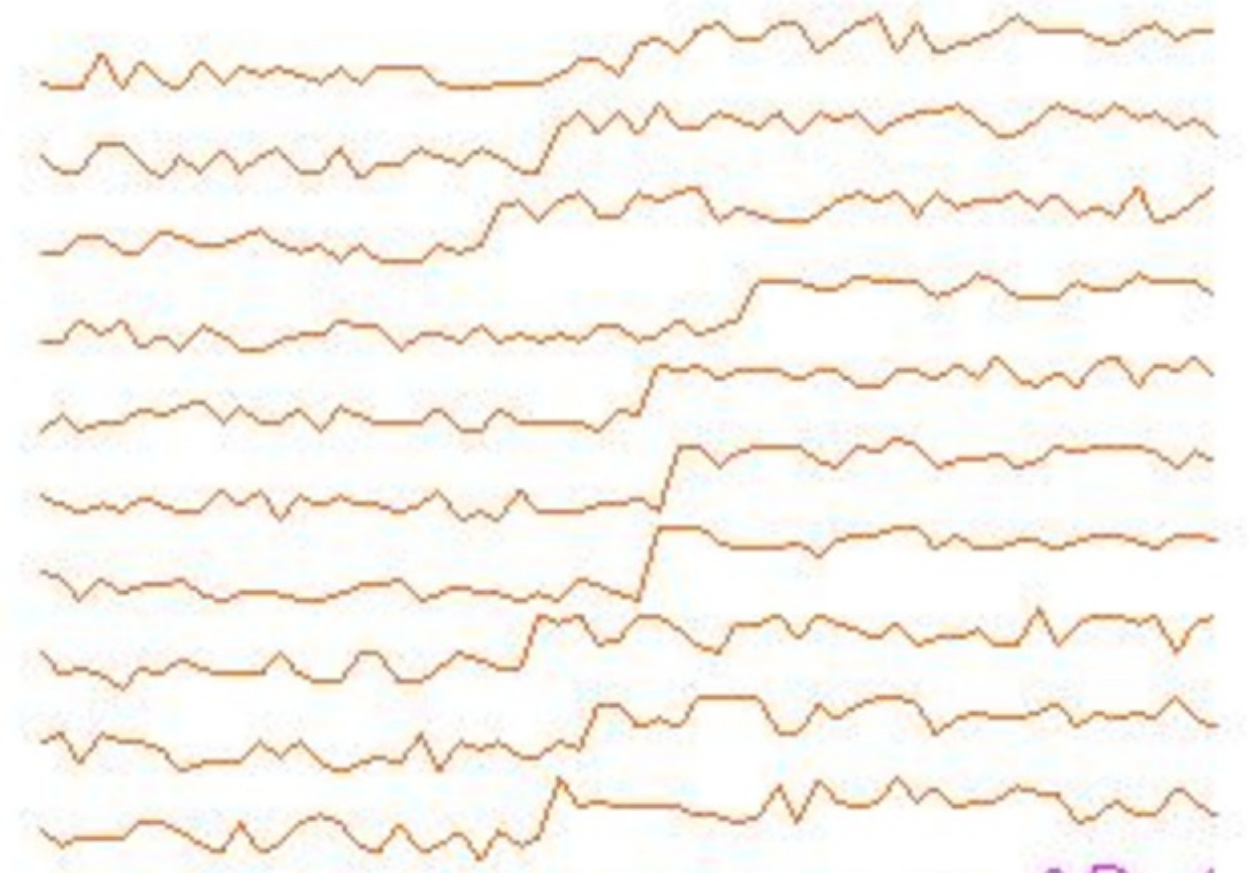
Increasing trend



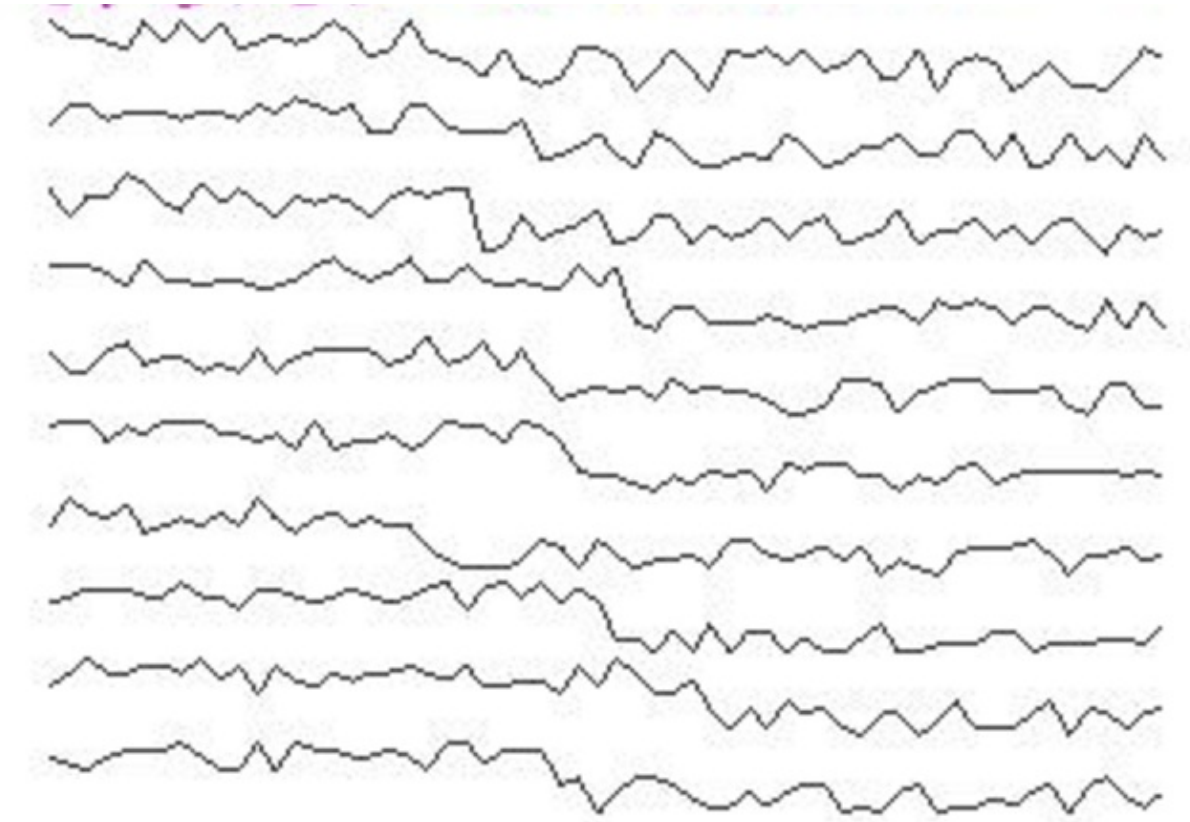
Decreasing trend



Upward shift



Downward shift



Lab Overview

- UCIdata.java
  - Utility class to download dataset
  - DO NOT EDIT UCIdata.java
- UCISequenceClassificationExample
  - The class you will edit for this lab

Lab Step 1

Navigate to the UCISequenceClassificationExample and add a line to call the UCIData class that downloads the data. ***Note*** the download takes time, once downloaded another call to UCIData verifies the download has occured and does NOT download again. It is safe to run the download twice.

Add this line to the class.

```
UCIData.download();
```

Lab Step 2

Set up the Record Readers for the Test and Train Data.

Add this line to the class.

```
SequenceRecordReader trainFeatures = new CSVSequenceRecordReader();
        trainFeatures.initialize(new NumberedFileInputSplit(ai.skymind.training.so
lutions.UCIData.featuresDirTrain.getAbsolutePath() + "/%d.csv", 0, 449));
        SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
        trainLabels.initialize(new NumberedFileInputSplit(ai.skymind.training.solu
tions.UCIData.labelsDirTrain.getAbsolutePath() + "/%d.csv", 0, 449));
```

Lab Step 3

Set the miniBatchSize and the number of classes.

Add these lines to the class.

```
int miniBatchSize = 10;
int numLabelClasses = 6;
```

Lab Step 4

Create a DataSetIterator for the training Data.

The JavaDoc for the needed class is here, https://deeplearning4j.org/datavecdoc/org/datavec/api/records/reader/impl/csv/CSVSequenceRecordReader.html

Add this line to the class.

```
DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(trainFeatures,
 trainLabels, miniBatchSize, numLabelClasses,false, SequenceRecordReaderDataSetIte
rator.AlignmentMode.ALIGN_END);
```

Lab Step 5

Normalize the data.

Add the following lines to the class.

```
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainData);
trainData.setPreProcessor(normalizer);
```

Lab Step 6

Set up the Test Data.

Add the following lines to the class.

```
 SequenceRecordReader testFeatures = new CSVSequenceRecordReader();
 testFeatures.initialize(new NumberedFileInputSplit(ai.skymind.training.solutions.
UCIData.featuresDirTest.getAbsolutePath() + "/%d.csv", 0, 149));
 SequenceRecordReader testLabels = new CSVSequenceRecordReader();
 testLabels.initialize(new NumberedFileInputSplit(UCIData.labelsDirTest.getAbsolut
ePath() + "/%d.csv", 0, 149));
 DataSetIterator testData = new SequenceRecordReaderDataSetIterator(testFeatures,
testLabels, miniBatchSize, numLabelClasses,false, SequenceRecordReaderDataSetItera
tor.AlignmentMode.ALIGN_END);
 testData.setPreProcessor(normalizer);
```

Lab Step 7

Set up the Neural Network

Add the following lines of code to the class.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(123)
    .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT).iteration
s(1)
    .weightInit(WeightInit.XAVIER)
    .updater(Updater.NESTEROVS).momentum(0.9)
    .learningRate(0.005)
    .gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
    .gradientNormalizationThreshold(0.5)
    .list()
        .layer(0, new GravesLSTM.Builder().activation("tanh").nIn(1).nOut(10).buil
d())
        .layer(1, new RnnOutputLayer.Builder(LossFunctions.LossFunction.MCXENT)
            .activation("softmax").nIn(10).nOut(numLabelClasses).build())
            .pretrain(false).backprop(true).build();

        MultiLayerNetwork net = new MultiLayerNetwork(conf);
        net.init();
```

Lab Step 8

Add a UI

Add the following lines to the class.

```
UIServer uiServer = UIServer.getInstance();
StatsStorage statsStorage = new InMemoryStatsStorage();
net.setListeners(new StatsListener(statsStorage));
uiServer.attach(statsStorage);
```

Lab Step 9

Train the network.

Add the following lines to the class.

```
int nEpochs = 40;
    String str = "Test set evaluation at epoch %d: Accuracy = %.2f, F1 = %.2f";
    for (int i = 0; i < nEpochs; i++) {
        net.fit(trainData);
        Evaluation evaluation = net.evaluate(testData);
        log.info(String.format(str, i, evaluation.accuracy(), evaluation.f1()));
        testData.reset();
        trainData.reset();
}
        log.info("----- Example Complete -----");
```

-