MultiLayerConfiguration code x 3 DNNs

1. 784 –1000 (ReLU) –10 (Softmax)

INDArray.class | MLPMnistSingleLayerExample.java | Activation.class | *MLPMnistTwoLayerExample.java

```
        //Get the DataSetIterators:
        DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize, true, rngSeed);
        DataSetIterator mnistTest = new MnistDataSetIterator(batchSize, false, rngSeed);


        log.info("Build model....");
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(rngSeed) //include a random seed for reproducibility
            .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT) // use stochastic gradient descent as an optimizat
            .iterations(1)
            .activation(Activation.RELU)
            .weightInit(WeightInit.XAVIER)
            .learningRate(rate) //specify the learning rate
            .updater(new Nesterovs(0.98))
            .regularization(true).l2(rate * 0.005) // regularize learning model
            .list()
            .layer(0, new DenseLayer.Builder() //create the first input layer.
                    .nIn(numRows * numColumns)
                    .nOut(784)
                    .build())
            .layer(1, new DenseLayer.Builder() //create the second input layer
                    .nIn(784)
                    .nOut(1000)
                    .build())
            .layer(2, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD) //create hidden layer
                    .activation(Activation.SOFTMAX)
                    .nIn(1000)
                    .nOut(outputNum)
                    .build())
            .pretrain(false).backprop(true) //use backpropagation to adjust weights
            .build();

        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(5));  //print the score with every iteration

        log.info("Train model....");
        for( int i=0; i<numEpochs; i++ ){
        log.info("Epoch " + i);
            model.fit(mnistTrain);
        }
```

2. 784 –50 (ReLU) –20 (ReLU) –10 (Softmax)

INDArray.class | MLPMnistSingleLayerExample.java | Activation.class | MLPMnistTwoLayerExample.java

```
        double rate = 0.0015; // learning rate

        //Get the DataSetIterators:
        DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize, true, rngSeed);
        DataSetIterator mnistTest = new MnistDataSetIterator(batchSize, false, rngSeed);


        log.info("Build model....");
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(rngSeed) //include a random seed for reproducibility
            .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT) // use stochastic gradient descent as an optimizat
            .iterations(1)
            .activation(Activation.RELU)
            .weightInit(WeightInit.XAVIER)
            .learningRate(rate) //specify the learning rate
            .updater(new Nesterovs(0.98))
            .regularization(true).l2(rate * 0.005) // regularize learning model
            .list()
            .layer(0, new DenseLayer.Builder() //create the first input layer.
                    .nIn(numRows * numColumns)
                    .nOut(784)
                    .build())
            .layer(1, new DenseLayer.Builder() //create the second input layer
                    .nIn(784)
                    .nOut(50)
                    .build())
            .layer(2, new DenseLayer.Builder() //create the first input layer.
                    .nIn(50)
                    .nOut(20)
                    .build())
            .layer(3, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD) //create hidden layer
                    .activation(Activation.SOFTMAX)
                    .nIn(20)
                    .nOut(outputNum)
                    .build())
            .pretrain(false).backprop(true) //use backpropagation to adjust weights
            .build();

        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(5));  //print the score with every iteration
```

3. 784 –(architecture of your choice) –10 (Softmax)

Architecture of my own choice is using ELU instead of RELU, 784-50(ELU)-20(ELU)-10(Softmax)

INDArray.class | MLPMnistSingleLayerExample.java | Activation.class | MLPMnistTwoLayerExample.java

```
        double rate = 0.0015; // learning rate

        //Get the DataSetIterators:
        DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize, true, rngSeed);
        DataSetIterator mnistTest = new MnistDataSetIterator(batchSize, false, rngSeed);


        log.info("Build model....");
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(rngSeed) //include a random seed for reproducibility
            .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT) // use stochastic gradient descent as an optimizat
            .iterations(1)
            .activation(Activation.ELU)
            .weightInit(WeightInit.XAVIER)
            .learningRate(rate) //specify the learning rate
            .updater(new Nesterovs(0.98))
            .regularization(true).l2(rate * 0.005) // regularize learning model
            .list()
            .layer(0, new DenseLayer.Builder() //create the first input layer.
                    .nIn(numRows * numColumns)
                    .nOut(784)
                    .build())
            .layer(1, new DenseLayer.Builder() //create the second input layer
                    .nIn(784)
                    .nOut(50)
                    .build())
            .layer(2, new DenseLayer.Builder() //create the second input layer
                    .nIn(50)
                    .nOut(20)
                    .build())
            .layer(3, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD) //create hidden layer
                    .activation(Activation.SOFTMAX)
                    .nIn(20)
                    .nOut(outputNum)
                    .build())
            .pretrain(false).backprop(true) //use backpropagation to adjust weights
            .build();

        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(5));  //print the score with every iteration
```

Program output x 3 DNNs

1. The output of 784 –1000 (ReLU) –10 (Softmax)

```
========================Scores========================================
 # of classes:    10
 Accuracy:        0.9812
 Precision:       0.9811
 Recall:          0.9811
 F1 Score:        0.9811
Precision, recall & F1: macro-averaged (equally weighted avg. of 10 classes)
======================================================================
o.d.e.f.m.MLPMnistTwoLayerExample - ****************Example finished********************
```

2. The output of 784 –50 (ReLU) –20 (ReLU) –10 (Softmax)

```
========================Scores========================================
 # of classes:    10
 Accuracy:        0.9834
 Precision:       0.9834
 Recall:          0.9833
 F1 Score:        0.9834
Precision, recall & F1: macro-averaged (equally weighted avg. of 10 classes)
======================================================================
o.d.e.f.m.MLPMnistTwoLayerExample - ****************Example finished********************
```

3. The output of 784 –(architecture of your choice) –10 (Softmax)

Architecture of my own choice is using ELU instead of RELU, 784-50(ELU)-20(ELU)-10(Softmax)

```
========================Scores========================================
 # of classes:    10
 Accuracy:        0.9735
 Precision:       0.9732
 Recall:          0.9734
 F1 Score:        0.9732
Precision, recall & F1: macro-averaged (equally weighted avg. of 10 classes)
======================================================================
o.d.e.f.m.MLPMnistTwoLayerExample - ****************Example finished********************
```

Classification performance x 3 DNNs

Let’s classify the performance of each three DNNs. We can evaluate the performance of a model via Accuracy, Precision, Recall and F1 Score. Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. If we have high accuracy then its model is good. Accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, parameters are need to evaluate the performance of these models. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate. Recall is the ratio of correctly predicted positive observations to the all observations in actual class. F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. So, if the accuracy, precision, recall and f1 score are high, it has a good performance.

Among the results of three DNNs, second has the highest performance. And then, performance is high in the first and third order. First result only use three layers and use ReLU function. ReLU is the activation function which has no saturation problem at all. So it is computationally efficient. Second result use four layers and use ReLU function. And its performance is better than first one. So, the more layers there are, the better the performance will be. Third result use four layers and use ELU function. Its number of layers are same as second one, but the result of both are different. So, in some case, ReLU function will be better than ELU to improve the model’s performance.

Conclusion

All classification tasks depend upon labeled datasets. So program will transfer its knowledge to the dataset in order for a neural. And the neural order to dataset to learn the correlation between labels and data.

By the same token, exposed to enough of the right data, deep learning is able to establish correlations between present events and future events. The future event is like the label in a sense. I think, deep learning doesn't necessarily care about time, or the fact that something hasn't happened yet. And deep learning can read a string of number and predict the number most likely to occur next. In other words, it can learn information like a person and predict based on what it had learned.

In deep learning, the number of neurons(in this case, layers)is change the result of accuracy, precision, recall, F1 score. In general, it seems that as the number of layers increases, the value of the result increases. The kind of activation function used in deep learning also affects the result value. There is many activation functions : CUBE, ELU, HARDSIGMOID, IDENTITIY, RELU, RRELU, SOFTMAX, and so on. It is important to use activation function that can achieve the best performance. In conclusion, by adding a variety of factors, we can derive close to 100 percent accuracy. We can train a neural network with zero domain knowledge of computer vision.