Code

file:///Users/lijiahao/Documents/UBC/CPEN_502/UBC-CPEN502/src/Assignment1/Layer.java

A 'Layer' class is implemented in this part for reuse. Neural network with different layers and neurons could be constructed easily by adding or deleting 'Layer' object instead of hard coding.

```java
package Assignment1;

import java.util.Random;

public class Layer {

    private static boolean BINARY = true;

    private Layer prev;
    private Layer next;
    private final int N;
    private double[] values;
    private double[] deltas;
    private double[][] currWeights;
    private double[][] prevWeights;
    private double[] requiredOutputs;

    private final double a = 0.0d;
    private final double b = 1.0d;

    public Layer(int numOfNeurons) {
        N = numOfNeurons;
        values = new double[N + 1];
        values[N] = 1.d;
        deltas = new double[N];
    }

    /**
     * Each layer has two pointers pointing to its previous and next layer.
Once the connection is made,
     * the 2d-array weights[][] (if this is not the output layer) or the 1d-
array requiredOutputs[] (if
     * this is the output layer) would be initialized.
     * @param prevLayer
     * @param nextLayer
     */
    public void connectTo(Layer prevLayer, Layer nextLayer) {
        prev = prevLayer;
        next = nextLayer;
        if (next != null) {
            currWeights = new double[N + 1][next.N];
            prevWeights = new double[N + 1][next.N];
            requiredOutputs = null;
        } else {
            requiredOutputs = new double[N];
            currWeights = null;
            prevWeights = null;
        }
    }

    public void setInputs(double[] inputs) {
        if (prev != null)
            return;
        if (inputs.length != N)
            throw new IllegalArgumentException();
        for (int i = 0; i < inputs.length; i++)
            values[i] = inputs[i];
    }

    public void setOutputs(double[] outputs) {
        if (next != null)
            return;
        if (outputs.length != N)
            throw new IllegalArgumentException();
        for (int i = 0; i < outputs.length; i++)
            requiredOutputs[i] = outputs[i];
    }

    public double[] getOutputs() { return values; }

    public int size() { return N; }

    public Layer getPrev() { return prev; }

    public Layer getNext() { return next; }

    public double[][] getWeights() { return currWeights; }

    public void setWeights(int i, int j, double weight) {
        currWeights[i][j] = weight;
    }

    public void setRandomWeights() {
        double lower = -0.5d;
        double upper = 0.5d;
        if (next == null)
            return;
        for (int j = 0; j < next.N; j++) {
            for (int i = 0; i < N + 1; i++) {
                Random random = new Random();
                currWeights[i][j] = random.nextDouble() * (upper - lower) +
lower;
                prevWeights[i][j] = currWeights[i][j];
            }
        }
    }

    public void setZeroWeights() {
        if (next == null)
            return;
        for (int j = 0; j < next.N; j++) {
            for (int i = 0; i < N + 1; i++)
                currWeights[i][j] = 0.0d;
        }
        prevWeights = currWeights.clone();
    }

    public double sigmoid(double x) {
        return 2 / (1 + Math.exp(-x)) - 1;
    }

    public double customSigmoid(double x) {
        if (BINARY)
            return (b - a) / (1 + Math.exp(-x)) + a;
        else
            return sigmoid(x);
    }

    public void forwardPropagate() {
        if (next == null)
            throw new NullPointerException("Cannot perform a forward-
propagation on an output layer.");
        for (int j = 0; j < next.N; j++) {
            next.values[j] = 0.0d;
            for (int i = 0; i < N + 1; i++)
                next.values[j] += currWeights[i][j] * values[i];
            next.values[j] = customSigmoid(next.values[j]);
        }
    }

    public void backwardPropagate(double momentumTerm, double learningRate) {
        if (prev == null)
            throw new NullPointerException("Cannot perform a backward-
propagation on an input layer.");
        if (next == null) {
            for (int i = 0; i < N; i++) {
                deltas[i] = 0.0d;
                double error = requiredOutputs[i] - values[i];
                double derivative;
                if (BINARY)
                    derivative = values[i] * (1 - values[i]);
                else
                    derivative = (values[i] + 1) * 0.5 * (1 - values[i]);
                deltas[i] = derivative * error;
            }
        } else {
            for (int i = 0; i < N; i++) {
                deltas[i] = 0.0d;
                double derivative;
                if (BINARY)
                    derivative = values[i] * (1 - values[i]);
                else
                    derivative = (values[i] + 1) * 0.5 * (1 - values[i]);
                for (int j = 0; j < next.N; j++)
                    deltas[i] += currWeights[i][j] * next.deltas[j];
                deltas[i] = deltas[i] * derivative;
            }
        }
        for (int j = 0; j < N; j++) {
            for (int i = 0; i < prev.N + 1; i++) {
                double deltaWeight = prev.currWeights[i][j] -
prev.prevWeights[i][j];
                prev.prevWeights[i][j] = prev.currWeights[i][j];
                prev.currWeights[i][j] += momentumTerm * deltaWeight +
learningRate * deltas[j] * prev.values[i];
            }
        }
    }

    public void flip() {
        BINARY = !BINARY;
    }

}
```

file:///Users/lijiahao/Documents/UBC/CPEN_502/UBC-CPEN502/src/Assignment1/NeuralNet.java

```
package Assignment1;

import Sarb.NeuralNetInterface;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
```

```java
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Scanner;

public class NeuralNet implements NeuralNetInterface {

    private final double learningRate;
    private final double momentumTerm;

    private Layer inputLayer;
    private Layer outputLayer;

    private ArrayList<String> errorList = new ArrayList<>();

    /**
     * @param argLearningRate
     * @param argMomentumTerm
     * @param layers is a vararg which must contain one input layer, one output
layer, and at least one hidden layer,
     *              otherwise an IllegalArgumentException would be thrown
     */
    public NeuralNet(double argLearningRate, double argMomentumTerm, Layer...
layers) {

        learningRate = argLearningRate;
        momentumTerm = argMomentumTerm;

        int N = layers.length;
        if (N < 3)
            throw new IllegalArgumentException("Illegal neural network
construction.");
        inputLayer = layers[0];
        outputLayer = layers[N - 1];
        for (int i = 0; i < N; i++) {
            if (i == 0)
                layers[i].connectTo(null, layers[i + 1]);
            else if (i == N - 1)
                layers[i].connectTo(layers[i - 1], null);
            else
                layers[i].connectTo(layers[i - 1], layers[i + 1]);
        }
    }

    public double sigmoid(double x) {
        /* defined in Layer */
        return 0.0d;
    }

    public double customSigmoid(double x) {
        /* defined in Layer */
        return 0.0d;
    }

    public void initializeWeights() {
        for (Layer currLayer = inputLayer; currLayer != outputLayer; currLayer
= currLayer.getNext())
            currLayer.setRandomWeights();
    }

    public void zeroWeights() {
        for (Layer currLayer = inputLayer; currLayer != outputLayer; currLayer
= currLayer.getNext())
            currLayer.setZeroWeights();
    }

    public void flip() {
        for (Layer currLayer = inputLayer; currLayer != null; currLayer =
currLayer.getNext())
            currLayer.flip();
    }

    public void setTrainingSet(double[] X, double[] y) {
        inputLayer.setInputs(X);
        outputLayer.setOutputs(y);
    }

    public void forwardPropagate() {
        for (Layer currLayer = inputLayer; currLayer != outputLayer; currLayer
= currLayer.getNext())
            currLayer.forwardPropagate();
    }

    public void backwardPropagate(double momentumTerm, double learningRate) {
        for (Layer currLayer = outputLayer; currLayer != inputLayer; currLayer
= currLayer.getPrev())
            currLayer.backwardPropagate(momentumTerm, learningRate);
    }

    public double outputFor(double[] X) {
        initializeWeights();
        inputLayer.setInputs(X);
        forwardPropagate();
        return outputLayer.getOutputs()[0];
    }

    public double train(double[] X, double argValue) {
        return argValue - outputFor(X);
    }
    public int train(double[][] X, double[][] y) {
        initializeWeights();
        double[] totalError = new double[outputLayer.size()];
        int epoch = 0;
        do {
            for (int i = 0; i < outputLayer.size(); i++)
                totalError[i] = 0;
            for (int i = 0; i < X.length; i++) {
                setTrainingSet(X[i], y[i]);
                forwardPropagate();
                for (int j = 0; j < outputLayer.size(); j++)
                    totalError[j] += Math.pow(y[i][j] -
outputLayer.getOutputs()[j], 2);
                backwardPropagate(momentumTerm, learningRate);
            }
            totalError[0] /= 2;
            errorList.add(Double.toString(totalError[0]));
            epoch++;
        } while (totalError[0] > 0.05d);
        return epoch;
    }


    public void save(File argFile) {
        try {
            StringBuilder stringBuilder = new StringBuilder();
            for (Layer currLayer = inputLayer; currLayer != outputLayer;
currLayer = currLayer.getNext()) {
                for (int i = 0; i < currLayer.getWeights().length; i++) {
                    for (int j = 0; j < currLayer.getWeights()[0].length; j++)
                        stringBuilder.append(currLayer.getWeights()[i][j] + "
");

                    stringBuilder.append("\n");
                }
            }
            Files.write(argFile.toPath(), stringBuilder.toString().getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void load(String argFileName) throws IOException {
        Scanner scanner = new Scanner(new BufferedReader(new
FileReader("./weights.txt")));
        for (Layer currLayer = inputLayer; currLayer != outputLayer; currLayer
= currLayer.getNext()) {
            for (int i = 0; i < currLayer.getWeights().length; i++) {
                String[] line = scanner.nextLine().trim().split(" ");
```

```
                    for (int j = 0; j < currLayer.getWeights()[0].length; j++)
                        currLayer.setWeights(i, j, Double.parseDouble(line[j]));
                }
            }
        }

        public void saveError() {
            try {
                Files.write(Paths.get("./error.text"), errorList);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
```

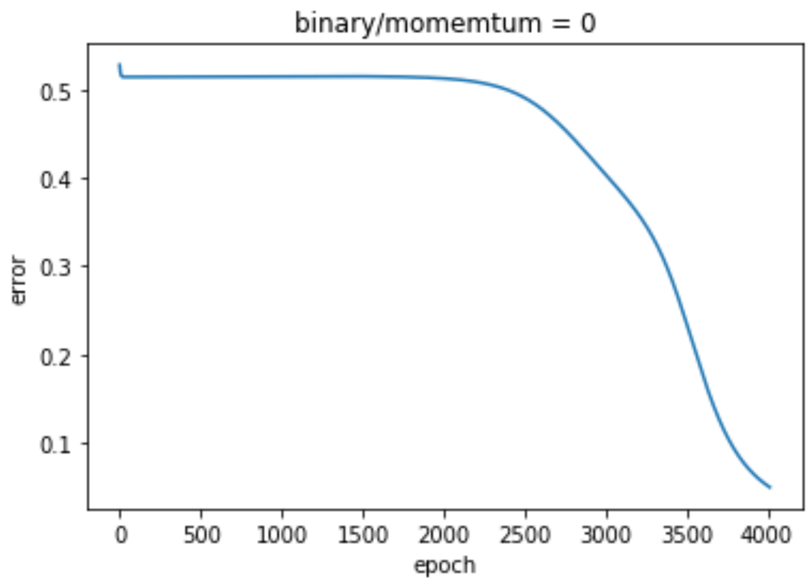
file:///Users/lijiahao/Documents/UBC/CPEN_502/UBC-CPEN502/src/Assignment1/OneLayerTest.java

This is the test for the one-hidden layer network. Layers are initialized first and converted into the varargs of the 'NeuralNet' object.

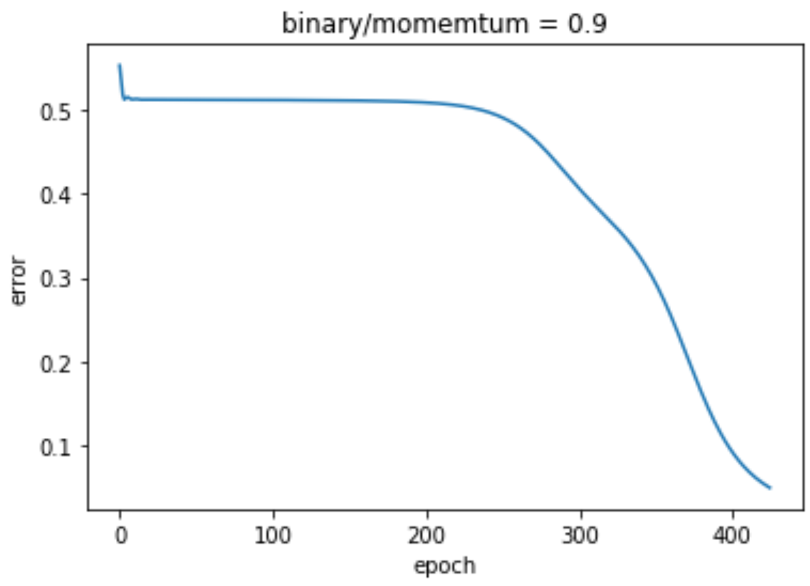
```
package Assignment1;

public class OneLayerTest {

    public static void binaryTest() {

        double[][] X = { {0, 0}, {0, 1}, {1, 0}, {1, 1} };
        double[][] y = { {0}, {1}, {1}, {0} };

        Layer inputLayer = new Layer(2);
        Layer hiddenLayer = new Layer(4);
        Layer outputLayer = new Layer(1);
        NeuralNet xor = new NeuralNet(0.2, 0, inputLayer, hiddenLayer,
outputLayer);

        int epoch = 0;
        for (int i = 0; i < 100; i++)
            epoch += xor.train(X, y);
        System.out.println(epoch / 100);
    }

    public static void bipolarTest() {

        double[][] X = { {-1, -1}, {-1, 1}, {1, -1}, {1, 1} };
        double[][] y = { {-1}, {1}, {1}, {-1} };

        Layer inputLayer = new Layer(2);
        Layer hiddenLayer = new Layer(4);
        Layer outputLayer = new Layer(1);
        NeuralNet xor = new NeuralNet(0.2, 0, inputLayer, hiddenLayer,
outputLayer);
        xor.flip();

        int epoch = 0;
        for (int i = 0; i < 100; i++)
            epoch += xor.train(X, y);
        System.out.println(epoch / 100);
    }

    public static void main(String[] args) {
        binaryTest();
        bipolarTest();
    }

}
```

Results

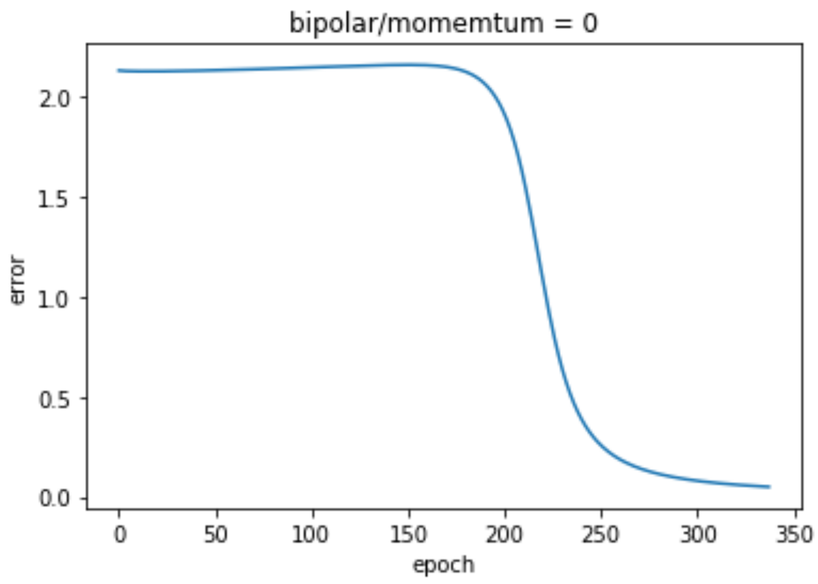
Below are the results for the one-hidden layer test:



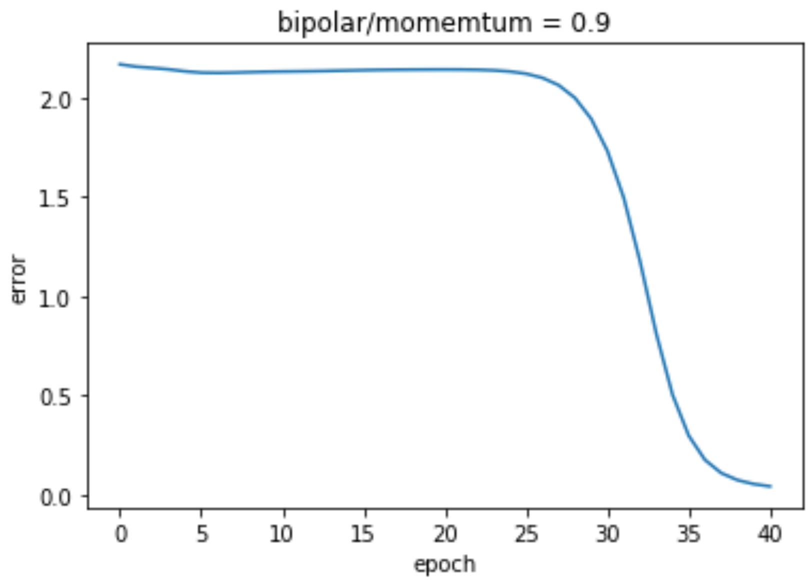
The average epoches for 100 tests are 4018.



The average epoches for 100 tests are 554.



The average epoches for 100 tests are 254.



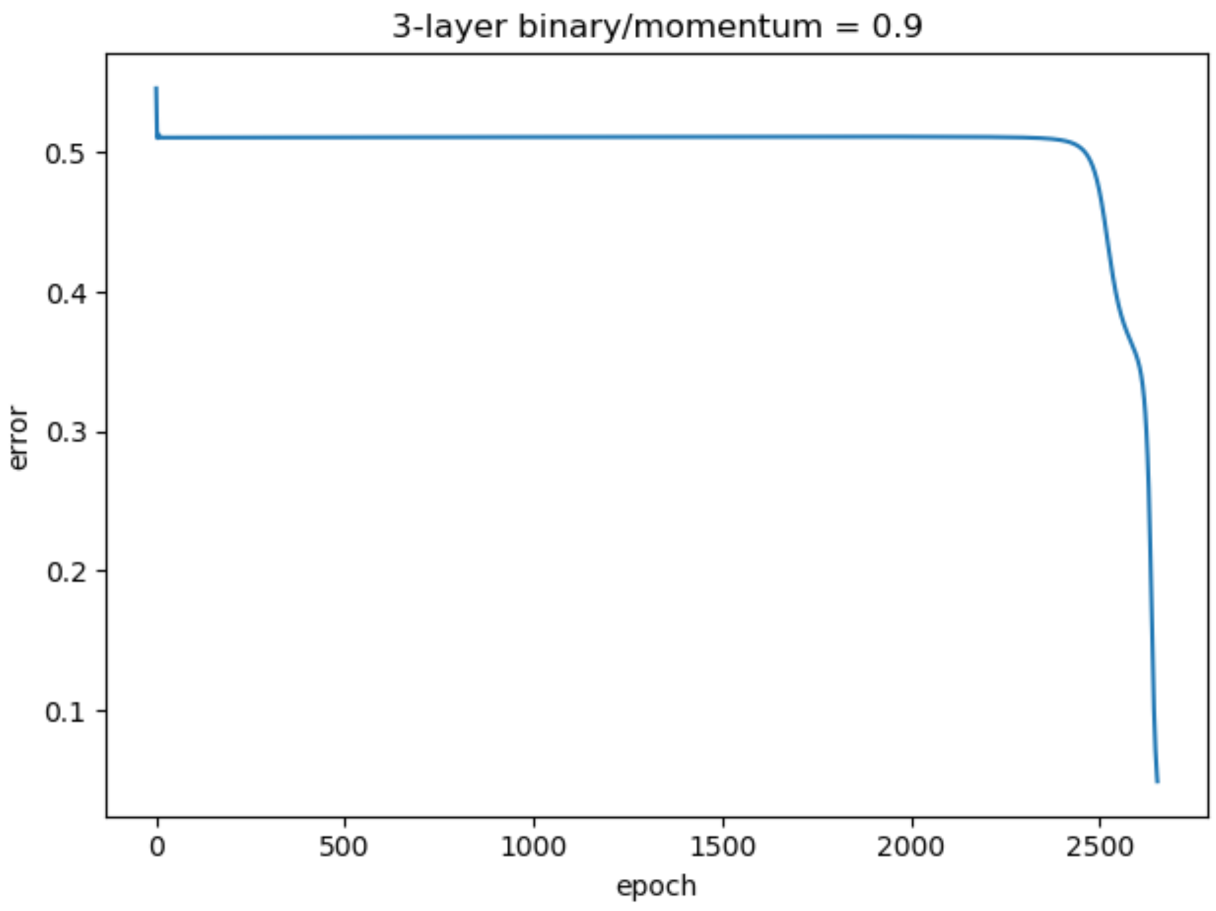
The average epoches for 100 tests are 45.

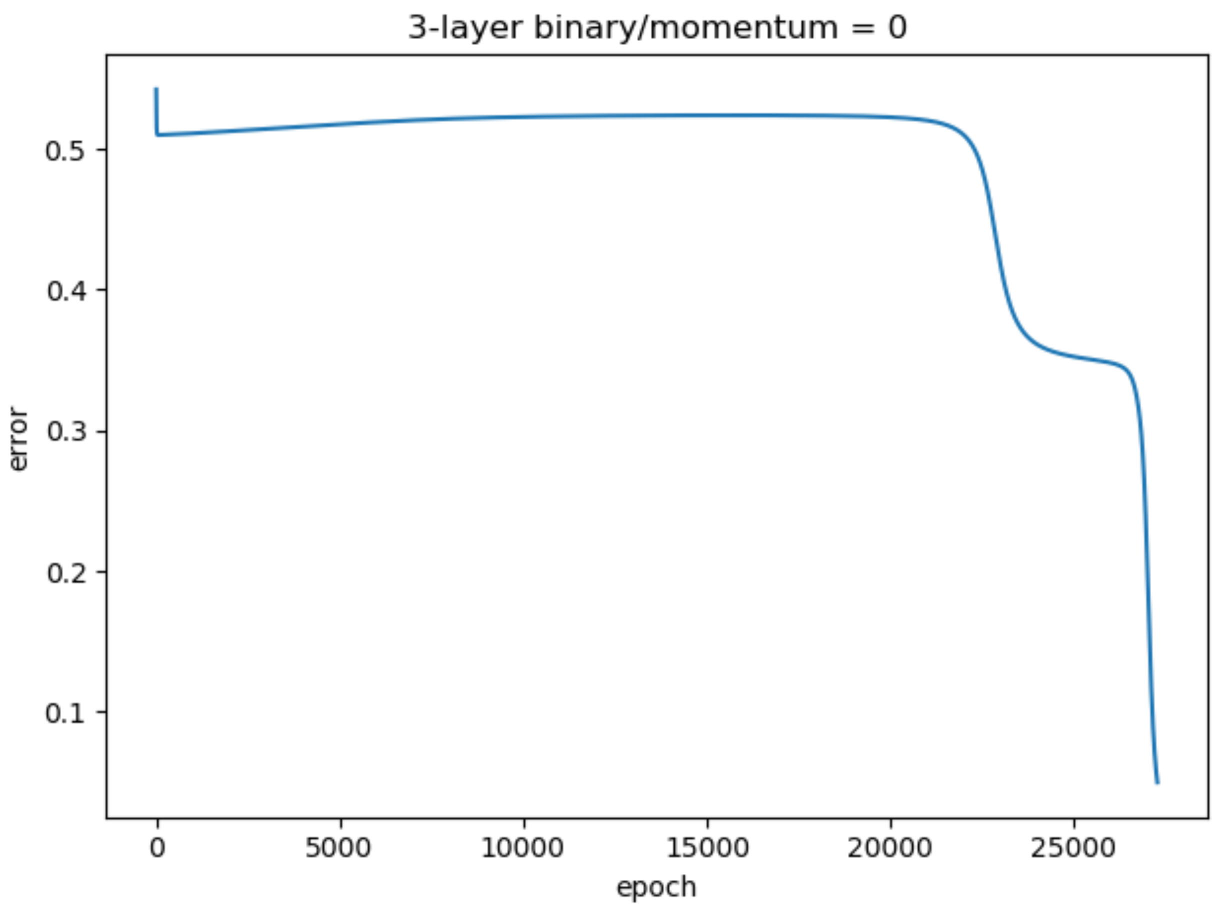
All the bipolar tests have one order of magnitude fewer epoches than their counterparts, the binary tests. The reason is that binary tests have 0 in the training set, which means sometimes update for the related weights is useless because any number times 0 equals 0.

After changing the momemtum from 0 to 0.9, the epoches that are necessary for the convergence reduce by 90% for both binary and bipolar tests.

For further study, a two-layer network is constructed with two hidden layers: one has 4 neurons and the other has 3 neurons:

```java
package Assignment1;

public class TwoLayerTest {

    public static void binaryTest() {

        double[][] X = { {0, 0}, {0, 1}, {1, 0}, {1, 1} };
        double[][] y = { {0}, {1}, {1}, {0} };

        Layer inputLayer = new Layer(2);
        Layer hiddenLayer1 = new Layer(4);
        Layer hiddenLayer2 = new Layer(3);
        Layer outputLayer = new Layer(1);
        NeuralNet xor = new NeuralNet(0.2, 0.9, inputLayer, hiddenLayer1,
hiddenLayer2, outputLayer);

        int epoch = 0;
        for (int i = 0; i < 100; i++) {
            epoch += xor.train(X, y);
        }
        System.out.println(epoch / 100);
    }

    public static void bipolarTest() {

        double[][] X = { {-1, -1}, {-1, 1}, {1, -1}, {1, 1} };
        double[][] y = { {-1}, {1}, {1}, {-1} };

        Layer inputLayer = new Layer(2);
        Layer hiddenLayer1 = new Layer(4);
        Layer hiddenLayer2 = new Layer(3);
        Layer outputLayer = new Layer(1);
        NeuralNet xor = new NeuralNet(0.2, 0.9, inputLayer, hiddenLayer1,
hiddenLayer2, outputLayer);
        xor.flip();
        int epoch = 0;
        for (int i = 0; i < 100; i++) {
            epoch += xor.train(X, y);
        }
        System.out.println(epoch / 100);
    }

    public static void main(String[] args) {
        binaryTest();
        bipolarTest();
    }

}
```





For a two-layer network, the number of epoches needed for the binary test is more than 2500 (momemtum = 0) and 25000 (momemtum = 0.9), much larger than their counterparts in the one-layer network. As for the bipolar test, the result cannot even reach its convergence point whatever the learning rate is. The reason may be that excessive layers make the network 'overfit' at each of the training set. Given that XOR problem is non-linearly separable, the hyperplane may not be able to 'find its position' with so many weights. However, I am still working on why the binary test has a better performance than the bipolar test in this network setting.