```java
package main;

import exception.NumberMismatchException;
import lombok.Getter;

import java.io.*;
import java.lang.reflect.Field;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

@Getter
public class NeuralNet implements NeuralNetInterface, Serializable {

    private int argNumInputs;
    private int argNumHidden;
    private double argLearningRate;
    private double argMomentumTerm;
    private double argA;
    private double argB;
    private List<Neuron> hiddenLayer;
    private Neuron outputNeuron;
    private boolean isBipolar;

    private static final double THREASHOLD = 0.05;


    /**
     * Constructor.
     * @param argNumInputs The number of inputs in your input vector
     * @param argNumHidden The number of hidden neurons in your hidden layer. Only a single hidden layer is supported
     * @param argLearningRate The learning rate coefficient
     * @param argMomentumTerm The momentum coefficient
     * @param argA Integer lower bound of sigmoid used by the output neuron only.
     * @param argB Integer upper bound of sigmoid used by the output neuron only.
     */
    public NeuralNet(int argNumInputs, int argNumHidden, double argLearningRate, double argMomentumTerm, double argA, double argB) {
        this.argNumInputs = argNumInputs;
        this.argNumHidden = argNumHidden;
        this.argLearningRate = argLearningRate;
        this.argMomentumTerm = argMomentumTerm;
        this.argA = argA;
        this.argB = argB;
        this.hiddenLayer = new ArrayList<>();
        this.isBipolar = argA + argB == 0;
        initializeWeights();
    }


    @Override
    public double outputFor(double[] X) {
        return forwardFeed(X);
    }

    @Override
    public double train(double[] X, double argValue) {
        return forwardFeed(X) - argValue;
    }

    public double[] setUpBias(double[] X){
        double[] temp = new double[X.length+1];
        System.arraycopy(X, 0, temp, 0, X.length);
        temp[temp.length - 1] = bias;
        return temp;
    }

    public int train(double[][] X, double[] targets) {
        int epoch = 0;
        double totalError;
        initializeWeights();
        StringBuilder stringBuilder = new StringBuilder();
        do {
            totalError = 0;
            for (int i = 0; i < X.length; i++) {
                double[] temp = setUpBias(X[i]);
                double yi = forwardFeed(temp);
                totalError += Math.pow(Math.abs(targets[i] - yi), 2) / 2;
                backProp(yi, targets[i]);
            }
            epoch++;
            stringBuilder.append(totalError + "\n");
        } while (totalError > THREASHOLD);
        String fileName = String.format("./data/%s_m%f_%s_%d.txt", isBipolar? "Bipolar" : "Binary",  this.argMomentumTerm, LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH-mm-ss")), epoch);
        File file = new File(fileName);
        try {
            file.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try (BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file))) {
            bufferedWriter.write(stringBuilder.toString());
            System.out.printf("The data is saved to file : %s \n", fileName);
            bufferedWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return epoch;
    }

    public double forwardFeed(double[] X) {
        List<Double> layer1Outputs = new ArrayList<>();
        try {
            for (Neuron neuron : this.hiddenLayer) {
                double curOutput = this.customSigmoid(neuron.sum(X));
                neuron.setOutput(curOutput);
                layer1Outputs.add(curOutput);
            }
            this.outputNeuron.setOutput(customSigmoid(this.outputNeuron.sum(layer1Outputs.stream().mapToDouble(i -> i).toArray())));
            return this.outputNeuron.getOutput();
        } catch (NumberMismatchException e) {
            System.exit(0);
        }
        return 0;
    }

    public void backProp(double yi, double target) {
        this.outputNeuron.setErrorSignalForOutputNeuron(target - yi);
        this.outputNeuron.updateWeights(this.argMomentumTerm, this.argLearningRate);
        for (int i = 0; i < this.hiddenLayer.size(); i++) {
            Neuron curNeuron = this.hiddenLayer.get(i);
            curNeuron.setErrorSignal(this.outputNeuron.getErrorSignal(), this.outputNeuron.getWeightByIndex(i));
            curNeuron.updateWeights(this.argMomentumTerm,
this.argLearningRate);
        }
    }

    @Override
    public void save(File argFile)  {
        try(ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream(argFile))) {
            objectOutputStream.writeObject(this);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void load(String argFileName) throws IOException {
        try(ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream(argFileName))) {
            NeuralNet neuralNet = (NeuralNet) objectInputStream.readObject();
            Class thisClass = this.getClass();
            for (Field field: neuralNet.getClass().getDeclaredFields()
                 ) {
                field.set(this, field.get(neuralNet));
            }
        } catch (ClassNotFoundException | IllegalAccessException e) {
            e.printStackTrace();
        }
    }

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

    @Override
    public double customSigmoid(double x) {
        return (this.argB - this.argA) / (1 + Math.exp(-x)) + this.argA;
    }

    @Override
    public void initializeWeights() {
        int neuronCount = this.argNumHidden + 1;
        this.hiddenLayer = new ArrayList<>();
        this.outputNeuron = new Neuron(this.argNumHidden
 + 1, this.isBipolar);
        while (neuronCount -- > 0) {
            hiddenLayer.add(new Neuron(this.argNumInputs
 + 1, this.isBipolar));
        }
    }

    @Override
    public void zeroWeights() {
        this.hiddenLayer.forEach(Neuron::zeroWeights);
        this.outputNeuron.zeroWeights();
    }
}

```