Part 1: OOP Concept

1. Objective

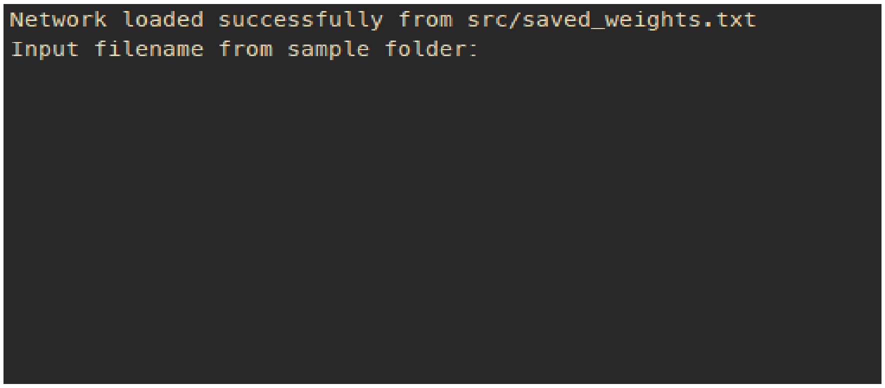
1) Be able to implement Objects and Classes.
2) Understand the concept of Encapsulation in OOP.

2. Instruction

1) Create a Java Project named "2110215_Midterm_Part1".
2) Copy all folders in "**toStudent/Part1_toStudent**" to your project directory **src** folder.
   a. Copy **saved_weights.txt** to your **project's src folder** also.
3) You are required to implement the following classes (details for each class is given in Sections 3 and 4.
   a. Neuron (package container)
   b. NeuronDemo (package container)
   c. Layer (package container)
4) JUnit for testing is in package test
5) Same UML as **umlQ1.png** in root folder of your project.
6) Export jar file (**q1.jar**) that includes your source code and your .class files and put it in **root** folder of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.

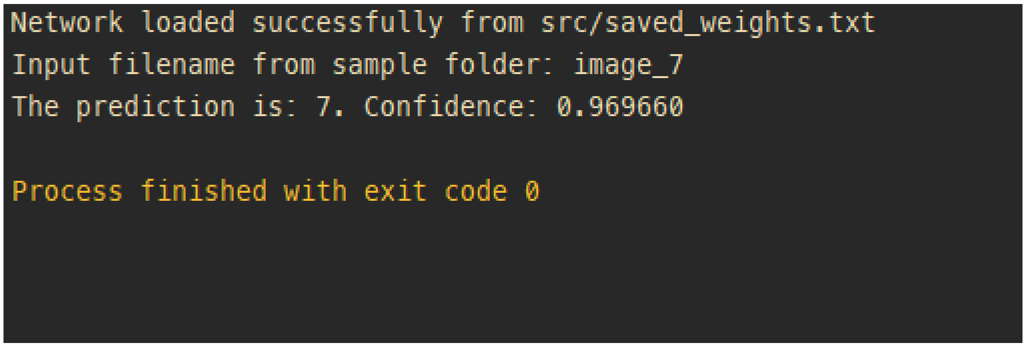
3. Problem Statement

You are implementing parts of a program that predicts a number in a picture. In this program, when a user runs the main class in the Main package, the user will be greeted with this.



```
Network loaded successfully from src/saved_weights.txt
Input filename from sample folder:
```

The /image folder contains 4 pictures to choose from. After inputting the file name that is in the “/image” folder, the image of that file will be loaded with the ImageProcessor class and the program will predict the number that appears in the image.



```
Network loaded successfully from src/saved_weights.txt
Input filename from sample folder: image_7
The prediction is: 7. Confidence: 0.969660

Process finished with exit code 0
```

The prediction uses neural network. The model is loaded with the data in the “src/saved_weights.txt” file using the implemented save/load method in the Network class.

4. Implementation Details

4.1) Package util

4.1.1) Class Activation

/ ALREADY PROVIDED */*

This class contains functions commonly used with neural networks.

Methods

Name	Description
+ double sigmoid(double x)	Computes the sigmoid activation function.
+ double tanh(double val)	Computes the hyperbolic tangent (tanh) activation function.
+ double relu(double val)	Computes the Rectified Linear Unit (ReLU) activation function.
+ double sigmoidDerivative(double x)	Computes the derivative of the sigmoid activation function.
+ double tanhDerivative(double val)	Computes the derivative of the hyperbolic tangent (tanh) activation function.
+ double reluDerivative(double val)	Computes the derivative of the Rectified Linear Unit (ReLU) activation function.

4.2) Package container

4.2.1) Class Neuron

You must implement this class from scratch.

This class represents a basic unit in a neural network, handling inputs, weights, bias, and output.

The class also includes methods to update weights and bias during training.

Fields

Name	Description
- double minWeightValue	min weight value of the neuron. This field is shared amongst all Neurons.
- double maxWeightValue	max weight value of the neuron. This field is shared amongst all Neurons.
- double[] weights	weights of the neuron. It is an array of double.
- double gradient	gradient of the neuron.

- double bias	bias of the neuron.
- double value	value of the neuron.

Constructors

Name	Description
+Neuron(double[] weights, double bias)	Create a new Neuron. Set the instance variables using the given parameters. Set the gradient and value to 0.
+Neuron(double value)	Create a new Neuron. Set the value according to the given parameter. Set bias and gradient to -1. Set weights to null.

Methods

Name	Description
+ void setRangeWeight(double min, double max)	This method will change the min weight value and max weight value using the given parameters.
+ getter setter for each appropriate variable	

4.2.2) Class NeuronDemo

You must implement this class from scratch.

This class is used just for quickly checking the Neuron.

Field

Name	Description
- Neuron n	A Neuron

Constructors (no constructor)

Methods

Name	Description
+public void demonstrate()	Set the instance variable n to be a Neuron with weights = {0.2,0.3,0.4} and bias equals to 0.2 Set the following properties of n: ● minimum weight value to 0.1 ● maximum weight value to 0.7 ● gradient to 0.6 ● value to 1
+getter setter for variable n	

4.2.2) Class Layer

You must implement this class from scratch.

This class represents a group of neurons. It provides functionality for initializing neurons with random weights and biases, as well as applying activation functions to the output of neurons within the layer.

Fields

Name	Description
- Neuron[] neurons	all neurons of this layer.

- Function function	Our “function” of this layer.

Constructors

Name	Description
+ Layer(int inNeurons, int nNeurons, Function function)	Create a new layer. Set the value of the function field according to the given parameter. Set neurons as a new array with length equal to nNeurons. For each neuron in neurons, prepare a weights array with the length equal to the value of *“inNeurons”*. Then, for each data in that weights array, set its value using ***“GenRandom.randomDouble”*** with Neurons’ minWeightValue and maxWeightValue. After finishing the weights array for each neuron, create the corresponding neuron object, with weights being the weights array and bias being the result of ***“GenRandom.randomDouble”*** on 0 and 1. Don’t forget to put the created Neuron into the Layer’s Neuron array. Hint: See GenRandom class to see the basic implementation of this utility. ***“GenRandom.randomDouble(double min, double max)”***

+ Layer(double[] input)	This constructor initializes the neurons array with the length equal to input’s length. Then, initialize each neuron in the array with the value from the same position in the input. For example, the 3rd Neuron (in neurons) is initialized using the 3rd value stored in the input array. Finally, set the “function” variable to null.

Methods

Name	Description
+ double applyActivation(double x)	This method applies the “function” with “x”. It returns the result from applying the function (use class Activation). First, check if the “function” is legal. Only SIGMOID, TANH, RELU are legal (see class Function). If the “function” is illegal, throw an IllegalArgumentException, with message "Unknown activation function: " + function. Otherwise, return the calculated result from applying the function.
+ double applyActivationDerivative(double x)	This method computes the derivative of the “function”, using x as input. It returns the result from applying the function derivative on x (use class Activation).

	First, check if the “function derivative” is legal. Only SIGMOID, TANH, RELU are legal (see class Function). If the “function” is illegal, throw an IllegalArgumentException, with message "Unknown activation function: " + function. Otherwise, return the calculated result from applying the function.
+ getter setter for each appropriate variables	

4.2.3) Class Data

/ ALREADY PROVIDED */*

This class represents a dataset, storing input data and corresponding outputs as arrays of double values. You do not need to know anything about this class.

4.2.4) Class Network

/ ALREADY PROVIDED */*

The Network class encapsulates the functionality for constructing and training a neural network, comprising layers of neurons. You do not need to know anything about this class.

4.3) package main

4.3.1) Class Main: The main class

This class is given, YOU MUST NOT IMPLEMENT THIS CLASS.

Name	Description
+ static void main(String[] args)	

5. Scoring Criteria (40 marks, will be scaled to 10)

Class Neuron 12 mark(s)

-	testConstructorWeighted	2 mark(s)
-	testConstructorValue	2 mark(s)
-	testWeightRange	1 mark(s)
-	testGetAndSetWeights	1 mark(s)
-	testGetAndSetBias	1 mark(s)
-	testGetAndSetValue	1 mark(s)
-	testGetAndSetGradient	1 mark(s)
-	testConstructorEmptyWeights	1 mark(s)
-	testConstructorNegativeBias	1 mark(s)
-	testConstructorNegativeValue	1 mark(s)

Class Layer 13 mark(s)

-	testConstructorWithFunction	5 mark(s)
-	testConstructorWithInput	2 mark(s)
-	testApplyActivationSigmoid	0.5 mark(s)
-	testApplyActivationException	1 mark(s)
-	testApplyActivationDerivativeSigmoid	0.5 mark(s)
-	testApplyDerivativeException	1 mark(s)
-	testApplyActivationTanh	0.5 mark(s)
-	testApplyActivationDerivativeTanh	0.5 mark(s)
-	testApplyActivationRelu	0.5 mark(s)
-	testApplyActivationDerivativeRelu	0.5 mark(s)
-	testSetNeurons	0.5 mark(s)
-	testSetFunction	0.5 mark(s)

Class NeuronDemo 5 marks

- testDemonstrate 5 mark(s)

* Full Score from Coding Section is **30 mark(s)**

UML of all implemented class **(png file only -> umlQ1.png) saved in your project root folder**

10 mark(s)

- Classes Neuron, Layer, NeuronDemo must be shown.
- class details must be correct.

The project folder must also contain .jar file,

- name it **q1.jar**.
- the .jar file must have the source code inside
- the .jar file must have the .class files
- OTHERWISE THIS QUESTION WILL NOT BE GRADED