BLG252E Object Oriented Programming Assignment 3

Release Date: May 5, 2020
Due Date: May 26, 2020, 23.59

Late submissions will NOT be accepted. Please submit your homeworks only through Ninova. This is an individual homework and in case of inappropriate exchange of information is detected, disciplinary actions will be taken.

- Your code must be compiled with following command: g++ 150150113.cpp
- Write your name and ID on the top of each document that you will upload.
- Compile your code on Linux using g++. Do not use any pre-compiled header files or STL commands. You can test your code through ssh. Be sure that you have included all of your header files.
- Use comments to explain your code.
- Be careful with the methods/attributes which are supposed to be constant, static, private.
- Handle exceptions and display error messages wherever necessary.
- If you have any questions about the homework please contact me via bektas18@itu.edu.tr
- Submit your homework in a .zip file where it contains all of the necessary .h and .cpp files.

Problem Description

In this homework, you are going to implement the simulation of a simple artificial neural network. You will only set the structure and implement a part of the system.

An artificial neural network is simply a system that is used to classify any data. It has an optimization process to do this but, since it is out of the scope of this homework, only the parts that will be used in this homework, such as the structure and feed forward process will be explained.

The diagram below represents the network used as an example for this homework. Your aim is to calculate “z” and “a” values in the diagram. Other parameters will be given to you.

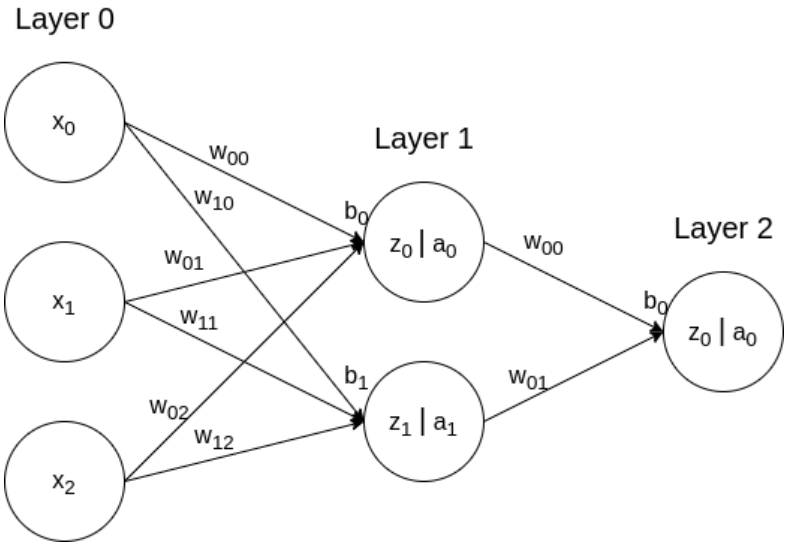
Calculations for layer 1:

$z_0 = w_{00} * x_0 + w_{01} * x_1 + w_{02} * x_2 + b_0$

$z_1 = w_{10} * x_0 + w_{11} * x_1 + w_{12} * x_2 + b_1$

$a_0 = f(z_0)$

$a_1 = f(z_1)$ (f is some activation function which will be explained)



If you think of x, b and z's as vectors and w as matrix -> X_{3x1}, B_{2x1}, Z_{2x1}, W_{2x3}; these operations will become a single matrix operation:
$Z_{2x1} = W_{2x3} * X_{3x1} + B_{2x1}$ (* is matrix multiplication)

For layer 2:
$Z_{1x1} = W_{1x2} * A_{2x1} + B_{1x1}$
$a_0 = f(z_0)$

Activation Functions:

In neural networks, some non-linear functions are applied after these linear matrix operations to give the model a non-linear characteristic. These functions are called "activation functions". Here are four of them you will use in this homework:

Sigmoid: $f(x) = 1 / (1 + e^{-x})$
ReLU: $f(x) = \max(0, x)$
Leaky ReLU: $f(x) = \max(0.1 * x, x)$

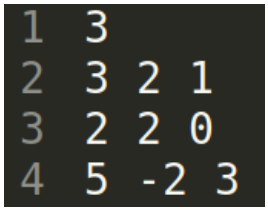
Class Definitions

- You need to implement a **Neuron** class.
  - This class will be an "**Abstract Base Class**"
  - It should have two attributes:
    - value (z), activated value (a)
  - It should have a **pure virtual method** called "activate":
    - virtual void activate() = 0;
- You need to implement a **SigmoidNeuron** class as a *subclass* of **Neuron** class**.**
  - It should have a method called "activate":
    - Apply sigmoid formula to it's value (z) and assign it to it's activated value (a).
- You need to implement a **TanhNeuron** class as a *subclass* of **Neuron** class**.**
  - It should have a method called "activate":

        - Apply tanh formula to it's value (z) and assign it to it's activated value (a).
- You need to implement a **ReluNeuron** class as a *subclass* of **Neuron** class**.**
    - It should have a method called "activate":
        - Apply relu formula to it's value (z) and assign it to it's activated value (a).
- You need to implement a **LReluNeuron** class as a *subclass* of **Neuron** class**.**
    - It should have a method called "activate":
        - Apply leaky relu formula to it's value (z) and assign it to it's activated value (a).
- You need to implement a **Layer** class**.**
    - This class should keep an array of "Neuron"s.
- You need to implement a **Network** class**.**
    - This class should keep an array of "Layer"s.

Workflow

1. Read the .txt file:



First line indicates how many layers to have.
Second line indicates how many neurons these layers have.
Third line indicates the type of the neurons at that layer:

- 0 for "Sigmoid"
- 1 for "Leaky ReLU"
- 2 for "ReLU"

Fourth line gives the input values (x from the diagram) for the network.
For this example:

- The network will have 3 layers.
- First layer will have 3 neurons and these neurons are of "ReluNeuron" class.
- Second layer will have 2 neurons and these neurons are of "ReluNeuron" class.
- Third layer will have 1 neuron and this neuron is of "SigmoidNeuron" class.

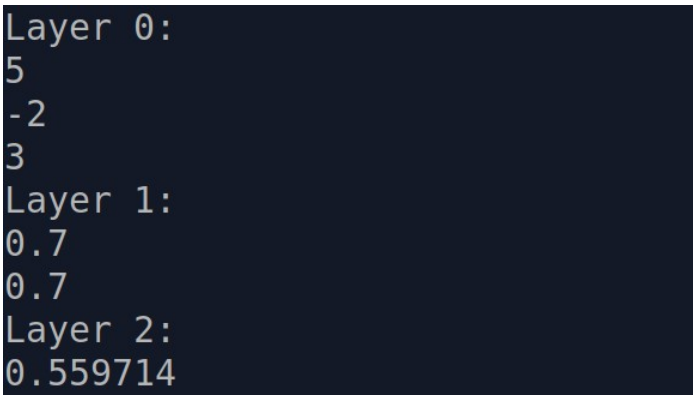
2. Set the parameters and construct the network:

- Automatically create the necessary W matrices and B vectors with proper shapes, fill them all with "0.1" value.
- The first layer (layer 0) is the input layer. This layer is different from others. You will **not** run any activation function in this layer. Set the neurons' **both** value and activated value as the input given in .txt file.

3. Calculate the values and activated values for each neuron except layer 0:

- Do matrix operations to calculate the values of the neurons in the next layer.
- Apply activation function -> activated value = f(value) (f is the corresponding activation function)

4. Print out the activated values of each neuron to the **console** in the following format:



Exceptions

For some cases you need to **throw** exceptions using **try** and **catch** blocks.

1. If given neuron type is invalid (out of 0-3 range) you need to throw: “Unidentified activation function!”
2. If the number of the input values does not match with the neuron size of layer 0, you need to throw: “Input shape does not match!”

A test file is provided for both cases.
You will only be graded upon these two exception cases.

Notes:

- If you use arrays, you should allocate memories dynamically.
- Make sure your code does not have any memory leaks.
- You should implement required Constructors, Destructors, getters and setters.
- Make use of constants, call by reference appropriately.
- Some sample inputs an expected outputs are provided. Your codes will be tested with other inputs as well.
- Read input files dynamically at execution time (eg. “./main input.txt”) (use argv[]).
- Print out your messages to **terminal**. Do **not** create a .txt file to print into.