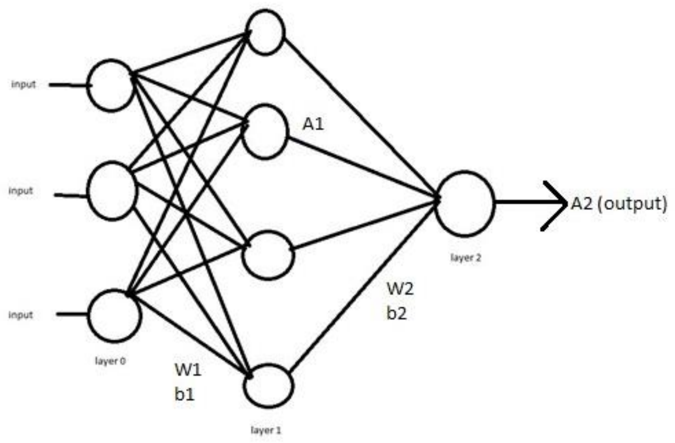
CS 426/525
Fall 2023
Project 1
Due: 13/11/2023

Background

Before explaining the assignment, let us give you some background information related to training neural networks in deep learning so that you may get an intuition for the significance of what you will implement.

Deep learning is a subset of machine learning that focuses on neural networks with multiple layers, also known as deep neural networks. The structure of the human brain inspires these deep neural networks and consists of many interconnected artificial neurons. Deep learning models are particularly effective for tasks such as image and speech recognition, natural language processing, and playing games. The two critical components of deep learning are **forward pass** and **backward pass**. Forward pass is the stage where the output of a neural network is calculated given the specified input. Backward pass, however, is calculating the gradients of the loss function with respect to the parameters of the network which can be used to update the parameters using gradient descent. The figure below shows a simple neural network that uses a sigmoid as an activation function.



- **sigmoid(x)** is defined as sigmoid(x) = $\frac{1}{1+e^{-x}}$
- **W$_\ell$** : Weight matrix of the layer ℓ
- **b$_\ell$** : bias vector of layer ℓ
- **X** : Input matrix. If X has dimensions of n x p, p represents the number of samples and n represents the input dimension.
- **A$_0$**: output of layer 0 which is same as X.
- **A$_1$**: output of layer 1. A_1 = sigmoid($W_1X + b_1$).
- **A$_2$**: output of layer 2 which is our prediction. A_2 = sigmoid($W_2A_1 + b_2$).

Forward pass is calculated as follows: You could think of the network as a pipeline where the output of layer ℓ is calculated using the output of layer (ℓ – 1). First, the output of layer 1 is computed, and after that, layer 2, layer 3, and so forth until the output of the last layer is calculated. In each layer, matrix multiplication of the corresponding weight matrix (W_ℓ) with the output of the previous layer ($A_{\ell-1}$) is calculated and summed with the bias vector b_ℓ. After that, the resultant matrix is fed into the sigmoid function elementwise to calculate A_ℓ.

Some useful links: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

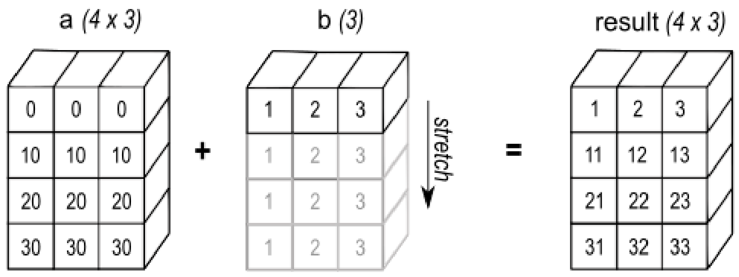
You do not have to fully understand what neural networks do and how they are implemented to do this assignment. Even though you may not understand it fully, you are advised to continue to read the assignment, as the assignment directives are self-explained and can be done without fully understanding neural networks.

Implementation of A Single Layer in a Fully Connected Network

The objective of the problem is to write a serial. A parallel program that **takes two matrices and one vector of float** as input, stored in an ASCII text file, and calculates the output = sigmoid(matrix.multiply(W,X) + b). The steps of calculation are given below:

Step 1: Calculate matrix multiplication of W and X and store the result in, say, variable **temp**. If the dimension of W is *m x n* and the size of X is *n x p*, then the resultant matrix (**temp** in our case) should have dimensions of *m x p*.

Step 2: Calculate temp2 = temp + b. In this case, temp has dimensions of *m x p,* and b has dimensions of *m x 1*. You may think that the dimensions do not match for matrix addition, but in many libraries designed for matrix multiplication and addition, if one of the matrices has less dimension than the other one (in our case, matrix has two dimensions where b is a column vector with one dimension) the matrix with lower dimensions is stretched to match dimensions with the other matrix. See the figure below, which shows how this is implemented in the **numpy** library.



https://numpy.org/doc/stable/user/basics.broadcasting.html

Therefore, you should logically or physically stretch the bias vector if the corresponding dimension does not match.

Step 3: Feed the matrix into the sigmoid function that calculates the **output = sigmoid(temp2)**. Here, you should apply function element-wise to each element of the matrix. In our case, temp2 has a dimension of *m x p*, so the output should also have *m x p* dimension.

Assignment

Part A - Serial Implementation (20 points):

In the serial implementation, you will have three matrices stored in the files weightmatrix.txt, input.txt, and bias.txt. W denotes the weight matrix, X denotes the input matrix, and b represents the bias matrix. Your serial program should read these three files, store them in corresponding arrays, calculate the result sigmoid(matrix.multiply(W,X) + b), and store it in result.txt.

Part B - Parallel Implementation (60 points):

In the parallel implementation, if the result matrix has, say, *m x p* dimensions, each process calculates *m / (number of processes)* rows of the resultant matrix.

Furthermore, in the parallel implementation, the master process reads the weightmatrix.txt, input.txt, and bias.txt files and stores them in corresponding arrays. Using point-to-point communication, you should send necessary data to other processes (you should decide which matrices or which part of matrices of W, X, and b are needed by which processes), and each process, including the master process, should compute *m / (number of processes)* rows of the result matrix. After that, using point-to-point communication, all processes except the master process should send the calculated rows of the result matrix to the master process. The master process should obtain the whole result matrix, store it, and then write it into result.txt.

Report (20 points):

Briefly explain your implementation. Plot graph(s) with various thread numbers for small, medium, and large input sizes indicating the performance of your implementation by comparing serial and parallel versions. What are your observations? Does parallel implementation improve the performance of any inputs? If not, what might be the reasons?

Example files (first row specifies the dimensions of matrix or vector):

```
$ cat input.txt
3 5
0.6 2.3 -3.6 7.5 4.2
1.2 -1.54 3.8 2.1 -5.3
-0.21 7.89 5.75 -1.5 2.5

$ cat bias.txt
4
3.1
2.1
-2.3
-0.6

$ cat weightmatrix.txt
4 3
5.6 -2.4 3.5
2.1 -0.6 2.89
3.7 -3.2 4.7
-2.25 -0.4 0.2
```

Submission

Submit a single zip file (**yourname_lastname_p1.zip**) to Moodle that includes:

- Your implementation with source files and necessary inputs for the following:
  - serial.c
  - parallel.c
  - readme.txt
- Run your code with various number of threads with various weight, input and bias matrices.
- Your report with 4 pages at most
- You are not allowed to use any built-in library for matrix multiplication, manipulation etc.
- You are not allowed to use collective communication features of MPI.
- You may assume the number of processes divides the number of the rows of the resultant matrix without any remainders.

Grading

Part A (serial): 20

Part B (parallel): 60

Report: 20

Email your questions to: yasir.altunhan@bilkent.edu.tr

Submission to: Respective Moodle assignment

Zip File name: yourname_lastname_p1.zip

No Late Submission Allowed!