Laboratory 5

(Due date: Nov. 2nd)

Objectives

- Compile and execute C++ code using the TBB library in Ubuntu 12.04.4 using the Terasic DE2i-150 Development Kit.
- Execute parallel applications using TBB: *parallel_for* and *parallel_reduce*
- Implement a Neural Network Layer with TBB.

Reference Material

- Refer to the board website or the Tutorial: Embedded Intel for User Manuals and Guides.
- Refer to the Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform → *Tutorial 6* for associated examples.

Activities

First Activity: Neural Network Layer Implementation in C++ with TBB (100/100)

- A sequential implementation of a Neural Network Layer was developed in *Laboratory 2*. In this activity, you are asked to parallelize the computation of the Neural Network Layer using *parallel_for* and *parallel_reduce*.
  - ✓ Though you are welcome to embed the parallel implementation of the neural network layer computation inside the functor you created in *Laboratory 2*, you are not required to do so.
  - ✓ For simplicity's sake, you can just read the input data and compute the output data with no need to specify a functor for the Neural Network Layer. You do need to create a functor for the specification of the operation of *parallel_reduce*.

NEURAL NETWORK

- A 3-layer neural network (also called a Fully Connected Layer) is depicted in Fig. 1(a). The input layer represents the input values to the network. Fig. 1(b) depicts the inputs and output of the first neuron (index '1') in layer 3.
- Fig. 1(c) depicts an artificial neuron model. The neuron output (action potential a_j^l) results from applying an activation function to the membrane potential (z_j^l). The indices correspond to the first neuron (index '1') in layer l.
- The membrane potential z_j^l is a dot product between the inputs and the associated weights, to which a bias is then added.

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l, l > 1$$

- The action potential intensity of a neuron is denoted by a_j^l, and it is modeled as a scalar function (activation function) of z_j^l:

$$a_j^l = \sigma(z_j^l) = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right), l > 1$$

  - ✓ Common activation functions include:
    - Rectified Linear Unit (ReLU): $\sigma(z_j^l) = \max(0, z_j^l)$
    - Hyperbolic Tangent: $\sigma(z_j^l) = \tanh(z_j^l)$
    - Sigmoid function: $\sigma(z_j^l) = 1/\left(1 + e^{-z_j^l}\right)$

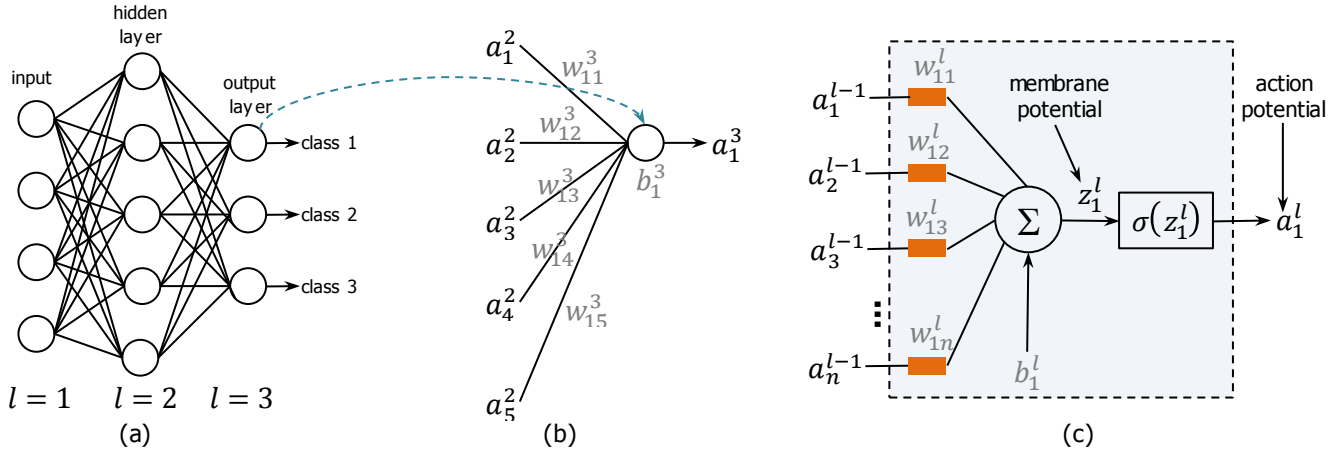

Figure 1. (a) 3-layer neural network. (b) First neuron (index '1') in layer *l*=3. (c) Artificial neuron model. The membrane potential is a sum of products (input activations by weights) to which a bias term is added. The neuron is the first neuron (index '1') in layer *l*. The input activations come from a previous layer (*l-1*).

- The output of a layer l can be described using a vectorized notation:

$$a^l = \sigma(z^l), \qquad z^l = w^l a^{l-1} + b^l, l > 1$$

where: w^l: weight matrix (NO rows by NI columns) of the layer l,
a^{l-1}: action potential vector (NI rows) of the previous layer l-1.
b^l: bias vector (NO rows) of the layer l.
z_l: membrane potential vector (NO rows) of the layer l.
a^l: action potential vector (NO rows) of the layer l.

✓ Fig. 2 depicts the matrix operation for z^l. NO: # of neurons in layer l. NI: # of inputs for each neuron in layer l.

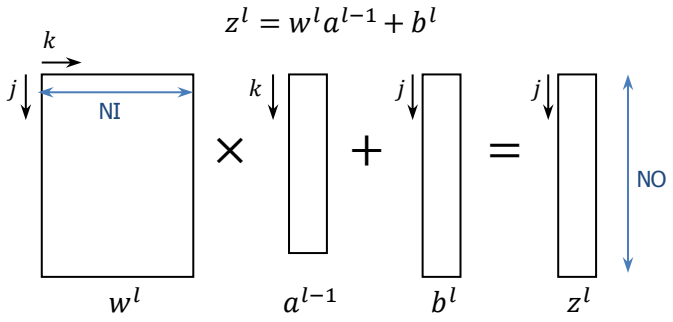

Figure 2. Matrix operation for the computation of all membrane potentials in layer l.

INSTRUCTIONS

- Write a C++ program (use `double` type for computations) that implements a neural network (NN) layer with TBB:
  - ✓ Neural network Layer:
    - Parameters: `NI, NO`. You can hard-code them in `main()`.
    - Input data: w^l, a^{l-1}, b^l, σ (activation function). Declare them in `main()` and then feed them to the NN layer.
    - Output data: z^l and a^l. Your code should print these values.
  - ✓ Fig. 3 depicts an example of an operation for NI=5, NO=4, Activation Function: ReLU.

$$\begin{bmatrix} 0.25 & 0.5 & -3.2 & -4.5 & -2.0 \\ 2.0 & 3.25 & 5.75 & 6.25 & 7.15 \\ 0.25 & -3.5 & 0.25 & 0.25 & 0.25 \\ 2.0 & 3.25 & 0.75 & -6.5 & 1.5 \end{bmatrix} \times \begin{bmatrix} 2.5 \\ 3.0 \\ 2.5 \\ 1.5 \\ 1.0 \end{bmatrix} + \begin{bmatrix} 2.0 \\ 1.5 \\ 2.5 \\ 3.5 \end{bmatrix} = \begin{bmatrix} -12.625 \\ 47.150 \\ -6.125 \\ 11.875 \end{bmatrix} \qquad \sigma \begin{bmatrix} -12.625 \\ 47.150 \\ -6.125 \\ 11.875 \end{bmatrix} = \begin{bmatrix} 0.000 \\ 47.150 \\ 0.000 \\ 11.875 \end{bmatrix}$$

w^l a^{l-1} b^l z^l z^l a^l

Figure 3. Testbed for your Neural Network Layer Implementation. NI=5, NO=4. Activation Function: ReLU.

  - ✓ **Strategy for parallelization with TBB**:
    - The following snippet computes z^l and a^l in a sequential fashion:

```
for (i = 0; i < NO; i++) {
  // Dot Product:
    z[i] = 0;
    for (j = 0; j < NI; j++) z[i] = z[i] + w[i][j]*a_i[j];

    z[i] = z[i] + b[i]; // membrane potential
    a_o[i] = act_fun(z[i],af); } // activation function (defined elsewhere)
```

      - Here, `a_i` represents a^{l-1}, `w` is w^l, `b` is b^l, and `af` is the activation function σ (entered as an integer). Also, `z` represents z^l and `a_o` represents a^l.
    - Use *parallel_for* to implement the for loop in the code snippet.
    - Use *parallel_reduce* to implement the dot product (whose result is `z[i]`).
      - This requires the implementation of a functor, the functor declaration, and the call to *parallel_reduce* that implements the operation (dot product) in the functor. See examples in *Tutorial 5*.
      - Recommendation: Implement the dot product as a function (otherwise you need to declare an array of functors and initialize them via the parameterized constructor). Inside the function, you declare the functor, execute *parallel_reduce* and return the result.
  - ✓ Verification: Use the values specified in Fig. 3. Your result (z^l, a^l) should match those listed in Fig. 3.

- Compile the code and execute the application on the DE2i-150 Board.
  - ✓ Example: `./my_nnlayer`

- Take a screenshot of the software running in the Terminal. It should show the resulting values of z^l and a^l.

SUBMISSION

- Demonstration: In this Lab 5, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Webex) with the instructor and demo it (using a camera).

- Submit to Moodle (an assignment will be created):
  - ✓ One .zip file:
    - 1st Activity: The .zip file must contain the source files (`.cpp, .h, Makefile`) and the requested screenshot.

TA signature: _____ **Date:** _____