Assignment 1- Task 1 Report

Name: Shubham Sarda Entry Number: 2018TT10958

In task 1, we've implemented simple functions used in a neural network such as matrix addition, multiplication, pooling, activation and probability calculation to facilitate complete neural network architectures in upcoming tasks.

First: "$ make" is used to compile the .cpp file(s) and create the executable(s). Makefile is used for this purpose. On running the "make" command, an executable "yourcode.out" is generated. We can also use "make clean" command to remove the executable file(s).

We'll briefly go over the different functions enabling different computations/subtasks. The following commands can be used to execute the 4 subtasks mentioned below.

1) ./yourcode.out fullyconnected inputmatrix.txt weightmatrix.txt biasmatrix.txt outputmatrix.txt

This command is used to compute output of a fully connected layer. A check is applied to verify if the number of arguments inputted are same as that required to execute the task (argc==6).

The input matrix, weight matrix and bias matrix are stored in 'inputmatrix.txt', 'weightmatrix.txt' and 'biasmatrix.txt' respectively. The arguments act as placeholders for actual path of files. The output matrix is stored in 'outputmatrix.txt'.

Output = Input * Weight + Bias, where, '*' = matrix multiplication, '+' = element wise addition.

Function: vector<vector<float>> fc_output(string input, string weights, string bias){…}

is used for computing output. Filenames of input, weight and bias matrices are passed as parameters to the function and used to read and create matrices in nested vector format. Helper function "read_matrix(..)" is used for this. Following this, helper functions "mul_matrix(..)" and "add_matrix(..)" are employed for matrix addition and multiplication of 2 matrices. A check is applied to verify if dimensions of input and weight matrices are of the form (A x B) and (B x C) respectively to ensure matrix multiplication is possible. Further dimensions of bias matrix are verified to ensure it is (A x C) for element wise addition to be possible. Finally, output matrix is written to the file 'outputmatrix.txt' in the required format (column major) using the helper "write_matrix(..)".

2) ./yourcode.out activation type inputmatrix.txt outputmatrix.txt

This command is used to compute output matrix after applying activation to the input matrix stored in 'inputmatrix.txt'. 'type' argument is a place holder for the type of activation applied. It can either be 'relu' or 'tanh'. In any other input to the 'type' argument, an error is printed on the console. Here as well, a check is applied to verify if the number of arguments inputted are same as that required to execute the task (argc==5).

Functions: a) vector<vector<float>> tanh(string input){..} b) vector<vector<float>> relu(string input){..}

These are the two function which apply tanh/relu activation element wise to the to matrix stored in the file with the name 'input' (passed as parameter to the function). Helper functions "tanh_float(..)" and

"relu_float(..)" are used to apply activations to single float value. Finally, output matrix is written to the file 'outputmatrix.txt' in the required format (column major).

3) ./yourcode.out pooling type inputmatrix.txt stride outputmatrix.txt

This command is used to compute output matrix after applying pooling to the input matrix stored in the file 'inputmatrix.txt'. 'type' argument is a place holder for the type of pooling applied. It can either be 'max' or 'average'. In any other input to the 'type' argument, an error is raised on the console. Here as well, a check is applied to verify if the number of arguments inputted are same as that required to execute the task (argc==6).

a) vector<vector<float>> average_pool(string input,int stride){..} b) vector<vector<float>> max_pool(string input,int stride){..}

The following functions are used to applying pooling on an input matrix stored in the file 'input'. The 'stride' is an integer argument which decides the stride and size of pooling filter. If stride=a, then (a x a) filter is used for pooling with a stride of 'a'. A check is applied to ensure that stride is an integer. It is also asserted that stride must divide the dimensions of input matrix. If not, an error is raised on the console. If the dimensions of input are (A x B), then dimension of output comes out to be (A/a x B/a). Maximizing/averaging is done on submatrices of size (a x a) in the input matrix to find individual elements of the output matrix. Finally, the output matrix is written to the file 'outputmatrix.txt' in the required format (column major).

4) ./yourcode.out probability type inputvector.txt outputvector.txt

This command is used to compute probability vector for a given input vector stored in the file 'inputvector.txt'. 'type' argument is a place holder for the type of probability being computed. It can either be 'softmax' or 'sigmoid'. In any other input to the 'type' argument, an error is raised on the console. Here as well, a check is applied to verify if the number of arguments inputted are same as that required to execute the task (argc==5).

a) vector<float> sigmoid(string file){..} b) vector<float> sigmoid(string file){..}

These functions are used to compute the probability vector. The file storing the input vector is passed as a parameter to the function. Helper functions "read_vector(..)" and "write_vector(..)" are utilized for reading of vector from the input file and writing the output vector to a new file respectively. The output vector is written to the file 'outputvector.txt' in the required format.

In the main function, it is also asserted that 2nd argument passed to the command line should be either of {"fullyconnected", "activation", "pooling", "probability"). In case its neither of these 4, an error message is raised on the console.

Thus, we're able to execute the required subtasks in a hassle-free manner. We haven't used assertions anywhere for error handling. In most cases, we try find the errors such as non-integer stride, incorrect number of command line arguments, incorrect argument etc. in the "main(..)" function itself using if-else statements. In case of an error in main, we raise the error on console and return 0 to smoothly exit

the function. Some errors are however caught during computation within internal functions. For instance, suitability of matrix dimensions for addition/multiplication is checked in the function: "fc_output". In case of an error, we print the required message on console and throw an exception to terminate the program smoothly so that the program doesn't go ahead with the wrong data. To give an example, the following message is displayed on console when bias matrix dimensions are not same as (input*weight) matrix (handled in "fc_output", hence exception is thrown).

```
The dimensions of input*weight matrix(2,1) doesn't match with the bias matrix(1,1)

Hence elements wise addition isn't possible.

terminate called after throwing an instance of 'std::exception'

  what():  std::exception

Aborted (core dumped)
```

In case the stride is not an integer, then the following message is printed on console (handled in main itself, hence no exception thrown, we just terminate the function with a return command):

```
Stride:'34.5' - given in arguments is not an integer.
```

Thus, appropriate error messages pop up on the console and help rectify the problem in case of errors.