Data Structure Assignment 1
利用”類神經網路”學習XOR的運算

通訊二 110503518 李秉宸

September 28, 2022

1 Compile 編譯結果

```
bingchen@LAPTOP-34JN212B:~/Data_Structure_Assignment1_Use_NN_to_learn_XOR$ gcc -o main main.c -lm layer.c neuron.c
bingchen@LAPTOP-34JN212B:~/Data_Structure_Assignment1_Use_NN_to_learn_XOR$
```

Figure 1: Compile file

2 Run 執行結果

Figure 2 to Figure 4 are the results of run file. When the users enter the "iteration times", the program will ask you to enter 1 or 2 to decide if you want to show the iteration process or not.

```
bingchen@LAPTOP-34JN212B:~/Data_Structure_Assignment1_Use_NN_to_learn_XOR$ ./main

Enter your target iteration times: 20000

Do you want to show the iteration process? Enter 1 if you want, and enter 2 if you don't.
```

Figure 2: Run file 1-1: Show the iteration process

If the user enter 1, the program will show the data just like Figure 3 (including iteration times, the training inputs and output results, sum of the square error and mean square error).

```
Iteration Times: 19998   Inputs/Output: 00/0  01/1  10/1  11/0  Error: 0.000069  MSE: 0.013419
Iteration Times: 19999   Inputs/Output: 00/0  01/1  10/1  11/0  Error: 0.000069  MSE: 0.013418
Iteration Times: 20000   Inputs/Output: 00/0  01/1  10/1  11/0  Error: 0.000069  MSE: 0.013418

Iterate successfully...
Final MSE: 0.013418
Now, you can enter your own test inputs (2~100 bits binary string, no space in the string)

Enter input to test: 1011
Output: 1

Enter input to test: 10111
Output: 0

Enter input to test: 0011010
Output: 1

Enter input to test:
```

Figure 3: Run file 1-2: Show the iteration process

If the user enter 2, it will show the data like Figure 4.

```
bingchen@LAPTOP-34JN212B:~/Data_Structure_Assignment1_Use_NN_to_learn_XOR$ ./main

Enter your target iteration times: 20000

Do you want to show the iteration process? Enter 1 if you want, and enter 2 if you don't.
2


Iterate successfully...
Final MSE: 0.010371
Now, you can enter your own test inputs (2~100 bits binary string, no space in the string)

Enter input to test:
```

Figure 4: Run file 2: Hide the iteration process

Otherwise, I set a enum called **"state"** to let the users decide the display mode by themselves. When the user **enter 1**, the program will set the state into **"SHOW_TRAINING"**, and into **"HIDE_TRAINING"** when the user **enter 2**.

After training, the program will set the state into **"TEST"** automatically, and let the user enter the inputs they want in a type of binary string between 2 to 100 bits.

```
15  enum state{
16      SHOW_TRAINING = 1,
17      HIDE_TRAINING = 2,
18      TEST = 3
19  };
```

Figure 5: The awy to show or hide the iteration process

3 Analysis Results 分析結果

3.1 Definition of Variables

To analysis the data, there are some variables of the program you show know:

"target_iteration_times" is decided by the user, and **"current_iteration_time"** is the real-time iteration times when the program is doing the training process.

"num_training_ex" means the example inputs used to train the program, and the value is 4 in default(inputs are 00 01 10 11 respectively).

"sum_squ_error" means the sum of the square error between desired output and actual output of all the training examples in every iteration. The program will just display **"Error"** to represent "sum_squ_error" when you run file.

"mse_total_error" means the sum of the average square error of all iteration. Note that this variable is not MSE, please see equation (2) to learn the difference. The program will just display **"MSE"** to represent "Mean-Square Error" when you run file.

3.2 Output File

To show the training process and results, after doing run file, the program will create a csv file in the folder of the project (the file name is **error_analysis.csv**), the file includes the iteration times, sum of the square error and mean-square error.

You can find the code about file processing in function "train_neural_net" in main.c.

If you want to see the whole data of the training, you can check out **"error_analysis(20220927).xlsx"** in the project folder.

Note that the data in the csv file will be cleared up in the next run file, and the data may not be the same to the result you get when you run file since the initial weight of training may be different.

3.3 Loss Function

Figure 6 and Figure 7 are the analysis results for "Iteration Times =20000", We can see the process of "Loss Convergence" from the charts. Some definition of the variables are in the following equations.

$$sum\_squ\_error = \sum_{i=1}^{num\_training\_ex} (Actual\ Output - Desired\ Output)^2 \tag{1}$$

$$MSE = \frac{\sum_{i=1}^{current\_iteration\_times} \frac{sum\_squ\_error}{num\_training\_ex}}{current\_iteration\_times} = \frac{mse\_total\_error}{current\_iteration\_times} \tag{2}$$

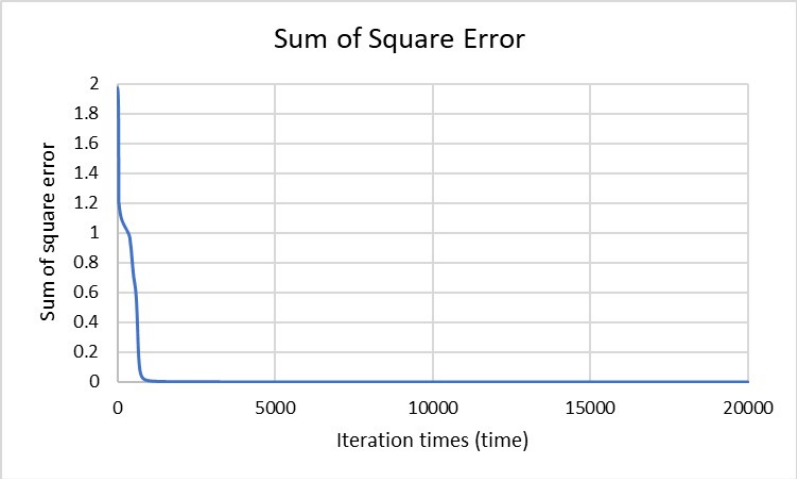

Figure 6: Result 1: Sum of Square Error (Target Iteration Times = 20000)

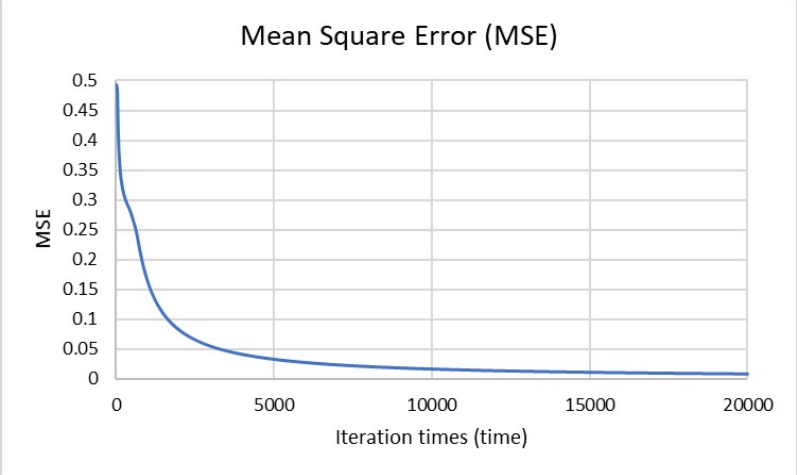

Figure 7: Result 2: Mean-Square Error (Target Iteration Times = 20000)

4 Improved 改善内容

4.1 Allocation Memories

I had used "malloc()" and "free()" in the program so that all the pointers and variables can be stored in allocation memories and free the memories after run file.
However, the pointer **"char *string"** is to be 100 bytes(100 char) as default, you can change the **"define INPUT_LENGTH 100"** into any other positive integer N you want, and the input string can store 2 to N bits after you change that.

Note: Actually, you can enter more bits than N, but it's not to be suggested.

4.2 N-bit Inputs

In function "test_nn" and "forward_prop" in main.c, I had used recursive function so that the program can test the n-bit inputs.
The input is a string, and the program will use ASCII code (char-'0') to check if it is a binary string.
Then, it'll calculate the output every 2-bit until the end of the string.

Here is the example:
Inputs: 01011
Step 1: Inputs 01(the first and second bit of the string), and output 1, check that it's not the end of the string.
Step 2: Inputs 10(the first 1 is from Step 1 output, and the other bit is the third bit of the string), and output 1, check that it's not the end of the string.
Step 3: Inputs 11(the first 1 is from Step 2 output, and the other bit is the forth bit of the string), and output 0, check that it's not the end of the string.
Step 4: Inputs 01(the first 0 is from Step 3 output, and the other bit is the last bit of the string), and output 1, check that it's the end of the string.
Step 5: Print the final output 1(from Step 4 output) on the screen.

5 Reference 参考資料

[1] GitHub：Neural-Network-framework-using-Backpropogation-in-C
(https://github.com/mayurbhole/Neural-Network-framework-using-Backpropogation-in-C.git)