Data Structure Homework #1
Binary Parity Checksum Validator with XOR Operator Trained by Neural Network

Ping-Chen,Chung(鍾秉辰)
Department of Communication Engineering
National Central University

September 27, 2022

Contents

Abstract

This report introduces the basic structure of neural networks and the C functions of dynamic memory allocation, which is used to precisely manage the memory of the program of neural network XOR operator training program. The program then implements the learnt result to calculate the binary parity checksum of an user-input string. The author then analyzed the graph of the loss function(MSE loss) and proposed the further improvement of this program.

1 Theoretical Background of Neural Network

In this section, we're going to introduce the definition of neural networks, and its implementations in C as a library.

1.1 Neurons

Neurons are the fundamental part of the neural network, and they simulate the real neuron from organisms. To turn neuron into a math model, Scientists has given the specific definition and functions of neurons.

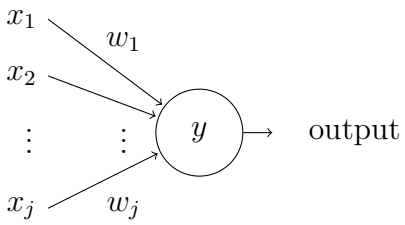

Figure 1: Model of Neuron [1]

Since the actual neuron in real life sends electric signals only if the inputs have reached specific conditions, we modeled it to be the *threshold* of the neuron to activate and send a signal to neurons in the next layer. *Weight* is then applied to every input to adjust the importance of each input, and the neuron only activates if the total sum of every input multiplied by its weight is greater than the threshold. The mathematical function of a neuron is given below:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{1}$$

Equation 1 shows one type of neurons: the perceptrons. However, in the modern model of neural networks, we use Sigmoid functions or Softmax functions

with other approximating techniques to evaluate neuron outputs to imporve the accuracy of the trained model.

1.2 Network of Neurons

As figure 2 shows, each neuron of the same layer is connected to every node of the next layer, therefore the output of this model can be transformed into matrix form:

Matrixlization of Neuron Networks [2]

Let x inputs be numbered $i_1, i_2, \cdots i_x$, then matrix $\mathbf{I}$ is the matrix storing $1 \times x$ elements of inputs in the same layer. Each input neuron has its y weights respectively points towards y neurons of the next layer, then the total of input weight is represented by an $x \times y$ matrix $\mathbf{W}$. Multiply $\mathbf{I}$ and $\mathbf{W}$ gets the inputs of the y neuron in the next layer, stored in $y \times 1$ matrix $\mathbf{H}$.
The same method can be done repetitively until the output layer is reached.

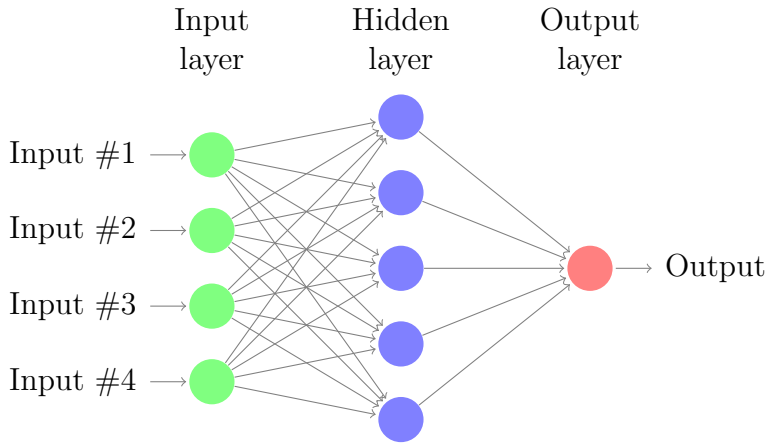

Figure 2: An Example of Neural Network

2 Techniques and Resources

In this section, we want to introduce a useful concept called *dynamic memory allocation*, and compare it with the known static memory allocation. We also mention the function library *genann* to use, for its simplicity to include.

2.1 Dynamic Memory Allocation in C

2.1.1 Difference of Static and Dynamic Memory Allocation

According to reference [3], the author has concluded the advantages and disadvantages of static memory allocation and dynamic memory allocation, shown below:

Static Memory Allocation

- Allocation of memory is done automatically by the compiler. This will reduce the execution time of the program.
- Uses stack data structures.
- Allocated memory (including variables and arrays) cannot be freed until the program terminates.
- Memory is allocated before the script executes.
- Simple usage and easy for new learners to use.

Dynamic Memory Allocation

- Memory is allocated in runtime. Hence, the execution time will be increased.
- Uses heap data structures.
- Memory can be resized/freed by scripts, therefore lessening wasted memory in the program.
- Memory should be freed manually after use, or it could cause unknown problems to debug.
- Needs better knowledge of memory allocation to prevent unknown errors.

2.1.2 Usable Functions to Dynamically Allocate Memory

- `malloc()` allocates the dynamic memory without the initialization.
- `calloc()` allocates the dynamic memory with initialization to prevent accidentally accessing corrupted memory.
- `realloc()` reallocates the existing memory to expand/shrink its memory size.
- `free()` frees the memory and let it be recycled by the operating system.

2.2 Function Libraries Used in This Program

In this program, we used an open-sourced neural network library Genann [4], with a small number of function library file to include and clear documentation for users to embed into their codes.

3 Program

This section contains the introduction of some parameters and self-defined functions to make users feel free reading source code.

3.1 Macro-Defined Constants

Name	Description	Default Value
ANN_INPUT_NUM	numbers of inputs in NN model	2
ANN_HIDDEN_LAYERS	numbers of hidden layers in NN model	1
ANN_NEURONS_PER_LAYER	neurons per layer in NN model	2
ANN_OUTPUT_NUM	numbers of inputs in NN model	1
ANN_LEARNING_RATE	the learning rate of neural network	3
LOOKUP_SIZE	temporary data storage for *genann*	4096
MAX_CHAR_LEN	maximum limit of user string input	100
ZERO_ASCII	value of 0 in ASCII	48
TRAIN_SETS	sets of neural network training	4

Table 1: Table of Macro-Defined Constants

3.2 Self-Defined Functions

void init();

This function initializes the space of the neural network's lookup size and array sample I/Os for later usage.

int selectMode();

This function asks the user to switch between functions by entering numbers corresponding to the functions by the instruction. The function will return an integer representing the enumeration code of functions.

char *getInput();

This function works when the user is asked to enter a string to show its checksum. The function will return the address of the stored string.

void nnQuadraticLoss(int iteration, int lossReprtSteps);

This function is used to train the model and display loss function (MSE) based on the given parameters:

- `int iteration` : the number of iteration times to train this model.

- `int lossReportSteps` : the interval of generating a report (a line of output in console) of loss.

For instance, if we use `nnQuadraticLoss(30000,1000);` , we will get 30000 times of model training and a report in every 1000 epochs.

void nnTrain(int iteration);
This function trains the model without showing the loss function.

void getBitString(char *string);
This function generates a string of bits based on the string user gives. The function will generate a string of bits and call the following function:

- **void getRealAns(char *string);** prints the binary parity checksum based on the pre-built XOR operator on the standard library.
- **void getTrainAns(char *string);** prints the binary parity checksum based on learned results.

void calloc2dArray(double **array, int row, int col);
This function creates a 2-dimensional array based on the given parameters:

- `double **array` : the address of the 2-dimensional array.
- `int row, int col` : the rows and columns of the array.

bool nnXor(char num1, char num2);
This function takes two characters (only 0 and 1 are allowed) and maps the inputs to the corresponding outputs from learned results.

3.3 Compile Parameters

Please open the project directory in command line and execute the following command:

```
$ make main
$ ./main
```

Listing 1: Compile Parameters

```
cc -Wall -Wshadow -O3 -g -march=native   -c -o main.o main.c
cc   main.o genann.o commonFunctions.o  -lm -o main
```

Listing 2: Compile Results

Note that the warnings from the compiler has been omitted for concise.

4 Execution Results

Below is one sample I/O of the program. Note that the input with >> is user input.

```
Welcome to this program. Please choose the mode you want to
    execute this program.
Enter 1 to show quadratic loss of the training process.
Enter 2 to enter custom string and show the XOR checksum
    based on NN learning output.
>>2
Please enter the training iteration. (integer)
The training result would be better if trained for 5000 times
     for 1 layer with 2 nodes per layer.
>>5000
Training Result: 0 1 1 0
If the result is not "0 1 1 0 ", please consider increase the
     iteration count or try rerun this program.
Please enter a string. This Program will output the XOR
    checksum of this string.
>>test_string

Checksum by XOR operator: 0
Checksum generated by neural network: 0
```

Listing 3: Main Program Execution Result

5 Analysis

5.1 Loss Function

From the reference [5], we can know that there exist various ways to evaluate the accuracy of a neural network. For simplicity, this program uses MSE (Mean-Square-Error) to be our loss function to evaluate the accuracy of the neural network. The equation is shown below:

$$\text{MSE} = (\frac{1}{n}) \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

where y_i is the actual training output and $\hat{y}_i$ is the desired output.

5.2 Graph Display

The graph below shows the loss function with sample training iterations (30000 times).

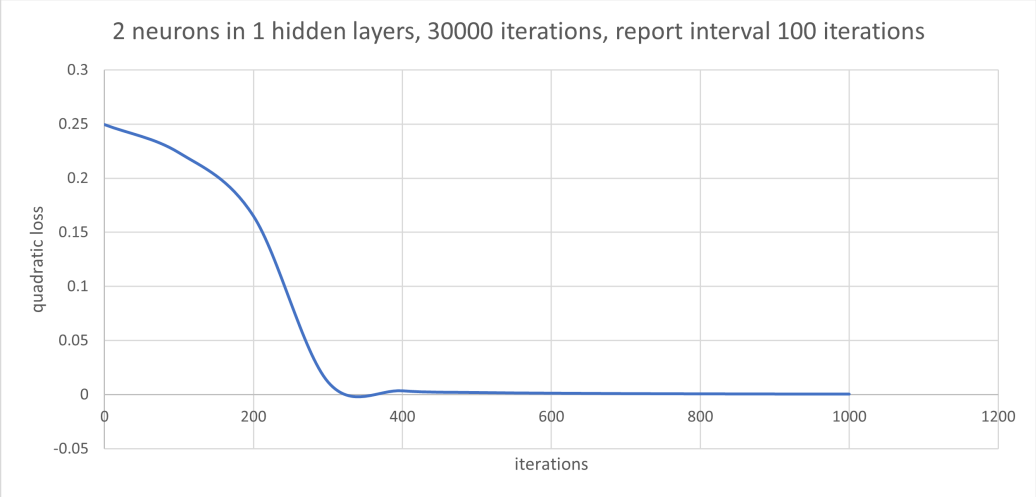

Figure 3: Graphs of Loss Function: MSE v.s. iterations

Note that this graph only shows up to 1000 iterations since the value after 1000 iterations approaches zero and is trivial. The graph at first reached its highest value (approximately 0.25), and then gradually falls down to zero with the increase of training epochs.

5.3 Further Improvement

Although the program has been successfully executed, there are still some issues to be fixed, listed below:

- a CSV output of the loss function could be added when training completes.
- Data storage of bits can be reduced to `bool` array instead of `char` array to reduce memory.
- In `void getBitString()` : the string of bits can be the function's return value. Doing this may clarify the workflow of `void getRealAns()` and `void getTrainAns()`.

References

[1] http://neuralnetworksanddeeplearning.com/chap1.html

[2] https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html

[3] https://www.geeksforgeeks.org/static-and-dynamic-memory-allocation-in-c/

[4] https://github.com/codeplea/genann

[5] https://www.statlect.com/glossary/loss-function/