NCU 2022

Data Structure

Neural Network Assignment

Cheng Yu Hin

110503534

Abstract

This project solves the problem of XOR checksum calculation using a neural network. It is written purely in C, and uses the open-source genann library for the neural network. The program trains the neural network using dataset provided in a text file, and improves the accuracy using feed-forward and backpropagation. The loss values during the training process is calculated using the Mean Absolute Error (MAE) function. It then, by using the trained neural network, evaluates a checksum value for a binary number entered by the user.

Table of Contents

The Problem

The main goal of this project is to evaluate the checksum of an n-bit binary value using XOR. The XOR operation must be implemented using a neural network.

In short, the project can be split into two parts:

1. Training a neural network to perform XOR operations
2. Evaluation of the checksum using the neural network
3. Without using matrices
4. Use dynamic memory allocation

The Design

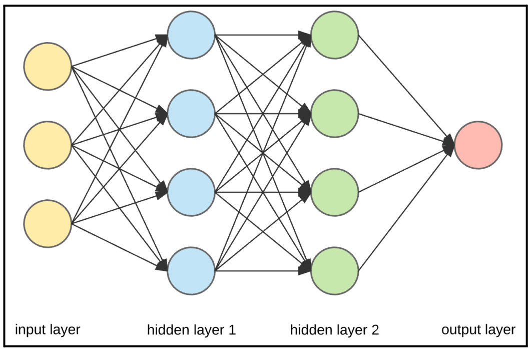
The neural network will be trained using feed-forward and backpropagation. It consists of one input layer, one output layer, and one or more hidden layers.

The network starts by randomly picking the weights for each node. It then obtains a predicted output based on the weights. The predicted output is then compared with the expected output given by us, and a loss value is obtained using the Mean-Absolute-Error (MAE) function. The MAE formula is given as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\, target\ output - predicted\ output \,|$$

By backpropagation, the network adjusts the weights of each node repeatedly based on the loss value obtained in each training iteration. The loss value will gradually approach 0 over time. The lower the loss value, the more accurate the network is.

The neural network can be visualized as follows:



A graphical representation of the a neural network [1]

[1] Image Classification with Convolutional Neural Networks. Ksenia Sorokina @Medium. Sep 27, 2022, https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

Implementation

The *genann* library is used for the neural network. It is an open-source neural network library written in C. It supports multi-layer neural networks with feed-forward and backpropagation enabled.

The training data (datasets) are provided in a text file (dataset.txt). The program will automatically retrieve the datasets from the file and store them in two arrays.

The datasets are then passed to the genann neural network for training. Through training, the network will learn to perform XOR operations.

After each training iteration, the network will be asked to predict an output. The predicted output will then be compared with the expected output, and a loss value is obtained using a loss function. The loss values allow us to see how accurately the network is doing.

After the network has been trained, the user will be asked to input a binary value. The program will perform XOR operations and calculate the checksum for this value using the trained neural network.

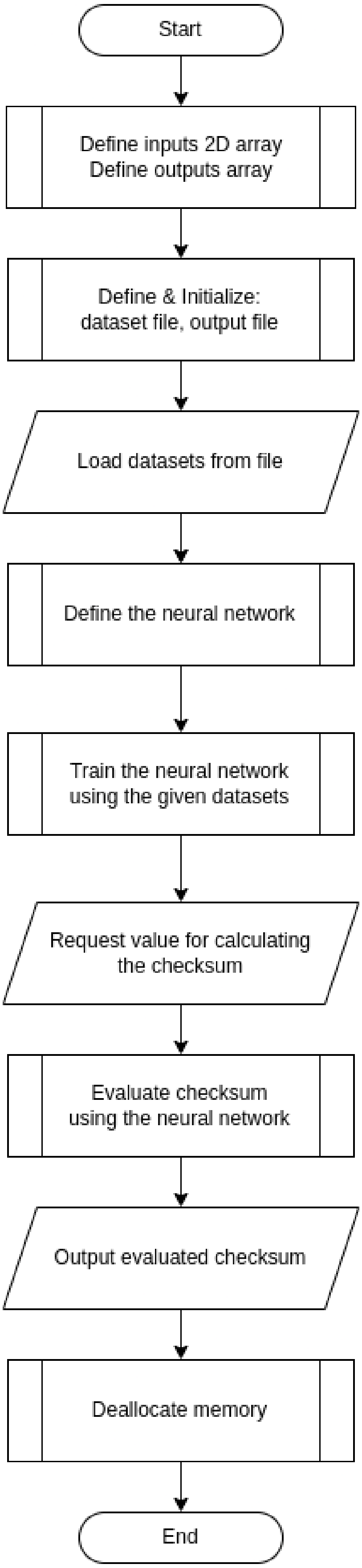
All the arrays are defined and initialized dynamically using the calloc() function, and are accessed directly using pointer values and offsets. Compared to the malloc() function, calloc() initializes the allocated memory bits to prevent accessing corrupted memory.

Below is the module structure of this project:

File	**Description**
main.c	Main program
checksum.c	Compute checksum using the trained neural network
fileops.c	File operations
training.c	Training the neural network

Each module, except for main.c, has a corresponding .h header file. These header files define the interface for external modules to access the functions within that module.

Program Flowchart



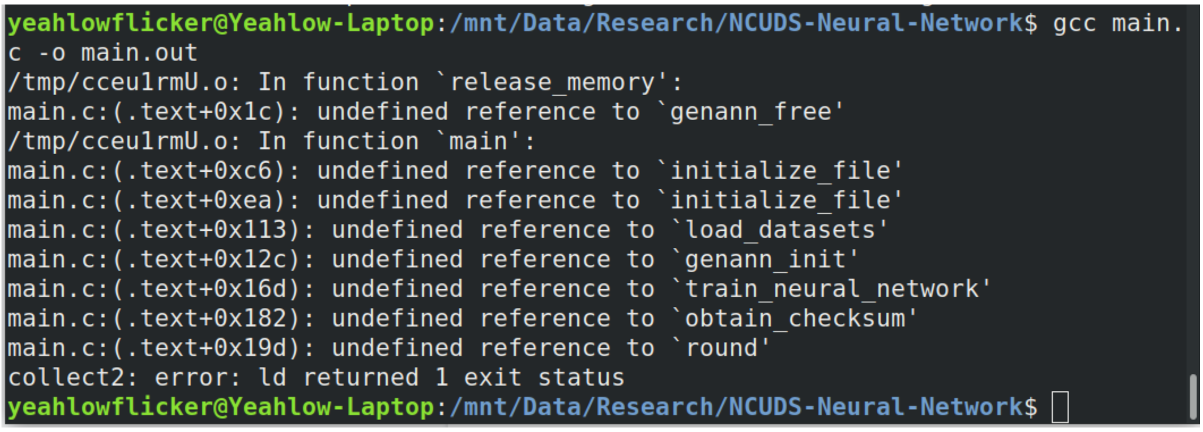
Compiling Results

The program can be compiled using the gcc module. On linux, one may compile using the following command:

```
gcc main.c -o main.out -lm fileops.c -lm training.c -lm checksum.c -lm genann.c -lm
```

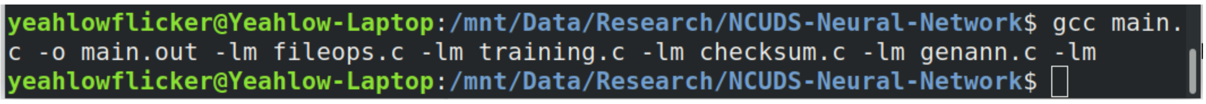
The `-lm` argument links the main libraries used by the program, while the `-o` argument specifies the output file name.

The xor.c and genann.c scripts must be compiled simultaneously in the same command, or otherwise the compiler will return an undefined reference error status as follows:



```
yeahlowflicker@Yeahlow-Laptop:/mnt/Data/Research/NCUDS-Neural-Network$ gcc main.
c -o main.out
/tmp/cceu1rmU.o: In function `release_memory':
main.c:(.text+0x1c): undefined reference to `genann_free'
/tmp/cceu1rmU.o: In function `main':
main.c:(.text+0xc6): undefined reference to `initialize_file'
main.c:(.text+0xea): undefined reference to `initialize_file'
main.c:(.text+0x113): undefined reference to `load_datasets'
main.c:(.text+0x12c): undefined reference to `genann_init'
main.c:(.text+0x16d): undefined reference to `train_neural_network'
main.c:(.text+0x182): undefined reference to `obtain_checksum'
main.c:(.text+0x19d): undefined reference to `round'
collect2: error: ld returned 1 exit status
yeahlowflicker@Yeahlow-Laptop:/mnt/Data/Research/NCUDS-Neural-Network$ 
```

When compiled successfully, the console output will be:



```
yeahlowflicker@Yeahlow-Laptop:/mnt/Data/Research/NCUDS-Neural-Network$ gcc main.
c -o main.out -lm fileops.c -lm training.c -lm checksum.c -lm genann.c -lm
yeahlowflicker@Yeahlow-Laptop:/mnt/Data/Research/NCUDS-Neural-Network$ 
```

A new file named “***main.c***” will be generated. To run the compiled application, use the following command (in the same directory):

```
./main.c
```

The Dataset and Output File

The dataset.txt file is required for the program to run. This file should contain a list of integer values used for XOR training. The expected inputs and outputs are provided in an order as follows:

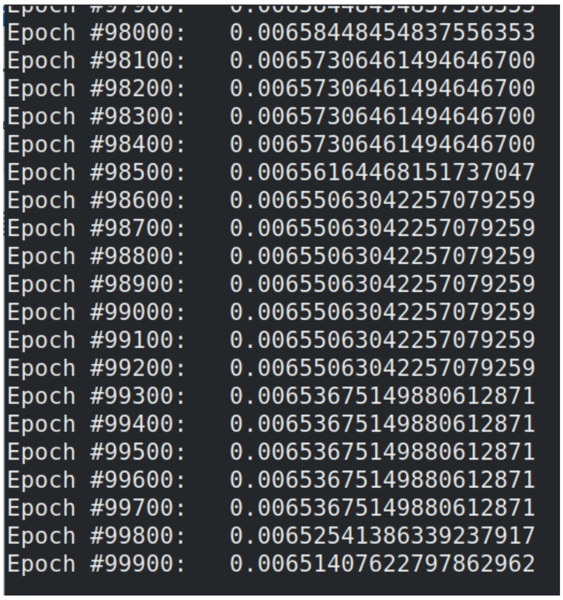
Input 0 of dataset 0
Input 1 of dataset 0
Output of dataset 0
Input 0 of dataset 1
Input 1 of dataset 1
Output of dataset 1
…
Input 0 of dataset 3
Input 1 of dataset 3
Output of dataset 3

To train a neural network to perform proper XOR operations, there should be at least 4 datasets. In total, there should be at least 12 lines of values representing the inputs and outputs.

After the program is run, the loss values of the training process will be written to the output.txt file. Each line contains a decimal value representing the loss value during each training iteration. This is useful for visualizing the loss.

Using the Program

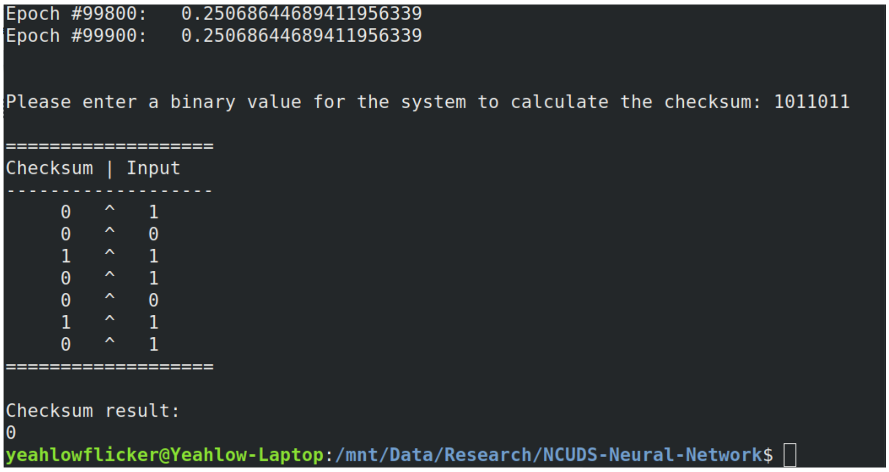
The program will automatically train the neural network. During each training iteration, the loss value obtained using the MAE function will be calculated. The loss value will be printed on the screen, and one may control the interval of the loss value printing by adjusting the LOG_ERROR_INTERVAL constant.



```
Epoch #98000:   0.00658448454837556353
Epoch #98100:   0.00657306461494646700
Epoch #98200:   0.00657306461494646700
Epoch #98300:   0.00657306461494646700
Epoch #98400:   0.00657306461494646700
Epoch #98500:   0.00656164468151737047
Epoch #98600:   0.00655063042257079259
Epoch #98700:   0.00655063042257079259
Epoch #98800:   0.00655063042257079259
Epoch #98900:   0.00655063042257079259
Epoch #99000:   0.00655063042257079259
Epoch #99100:   0.00655063042257079259
Epoch #99200:   0.00655063042257079259
Epoch #99300:   0.00653675149880612871
Epoch #99400:   0.00653675149880612871
Epoch #99500:   0.00653675149880612871
Epoch #99600:   0.00653675149880612871
Epoch #99700:   0.00653675149880612871
Epoch #99800:   0.00652541386339237917
Epoch #99900:   0.00651407622797862962
```

Outputs during the training process (Logging interval: 100)

After training, the user will be asked to input a binary value for calculating the checksum. Once the value is received, the program will calculate the checksum. Each step of the XOR checksum calculation will be printed:



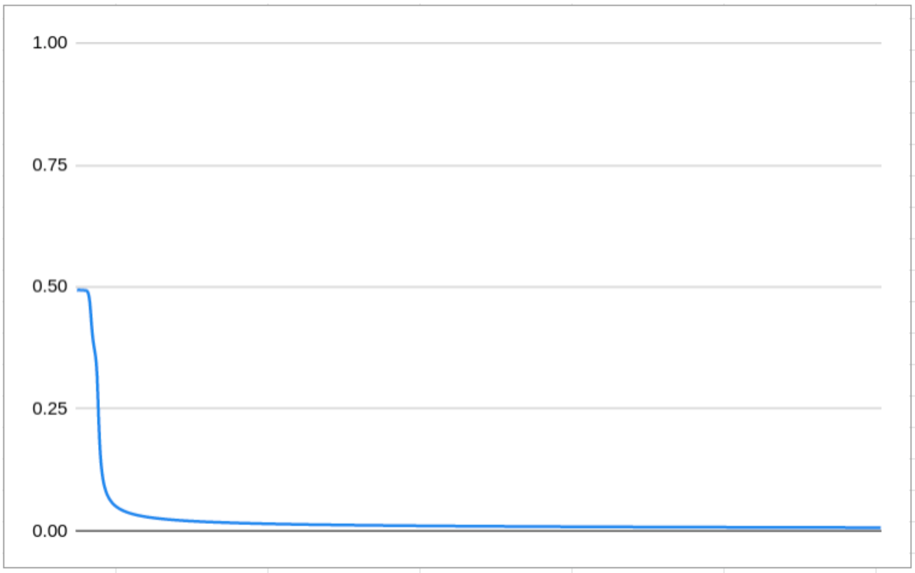
```
Epoch #99800:   0.25068644689411956339
Epoch #99900:   0.25068644689411956339


Please enter a binary value for the system to calculate the checksum: 1011011

===================
Checksum | Input
-------------------
     0   ^   1
     0   ^   0
     1   ^   1
     0   ^   1
     0   ^   0
     1   ^   1
     0   ^   1
===================

Checksum result:
0
yeahlowflicker@Yeahlow-Laptop:/mnt/Data/Research/NCUDS-Neural-Network$ 
```

Requesting user input; Outputs of the checksum calculation

Result Analysis

The loss value obtained throughout the training process showed a trend of gradient descent. Using the Mean Absolute Error (MAE) loss function, the initial cost at the beginning is about 0.5. As the network gets trained, the cost reduces gradually and reaches about 0.006. The reduction of the loss value indicates the improvement in the accuracy of the neural network.

Below is a visualization of the loss:



(Training Iterations: 100000, Learning Rate: 0.3, MAE)

And the corresponding loss outputs throughout the training process:

Epoch	Loss
#0	0.49373492281193187559
#100	0.49359322734938537725
#200	0.49405098897490828946
...	
#99700	0.00653675149880612871
#99800	0.00652541386339237917
#99900	0.00651407622797862962

Code Documentation

Macro-defined Constants

Name	Description	Default Value
SINGLE_INPUT_SIZE	The size of a single dataset input	2
SINGLE_OUTPUT_SIZE	The size of a single dataset output	1
DATASET_COUNT	How many given datasets are there	4
HIDDEN_LAYER_COUNT	How many hidden layers to define	1
SINGLE_HIDDEN_LAYER_NEURON_COUNT	How many neurons should each hidden layer has	2
TRAINING_ITERATIONS	How many times to train the neural network	100000
LEARNING_RATE	The learning rate of the neural network	3.0
LOG_ERROR_INTERVAL	How often to log the error of the training process	100
MAX_TEST_INPUT_LENGTH	How long can the binary input value be (for calculating the checksum)	100

Module: fileops.c

```
FILE* initialize_file(FILE *file, char* fileName, char* fileMode)
```

Open a file, assign its reference, and assert the file’s existence.

Parameter	Description
FILE* file	The pointer to the file
char* fileName	The name of the target file to be opened
char* fileMode	Which mode to use when opening the file

Returns: A pointer to the file reference

```
void load_datasets(FILE* datasetFile, double** inputs, double* outputs,
int dataset_count, int single_input_size)
```

Retrieve datasets from the dataset file, then assign the values retrieved to the inputs and outputs array correspondingly.

Parameter	Description
FILE* datasetFile	The pointer to the dataset file
double* inputs	The inputs array which the input data should be stored in
double* outputs	The output array which the output data should be stored in
`int dataset_count`	The total number of datasets
`int single_input_size`	The number of items in the input array of a single dataset

Returns: This function does not return any value.

Module: training.c

double mean_abs_error(double expected_output, **double** predicted_output**)**

Calculate the Mean Absolute Error (MAE).

Parameter	Description
double expected_input	The expected output value, provided by the dataset
double predicted_output	The predicted output, calculated by the neural network

Returns: A double value representing the resulting MAE.

---

double train_single_neuron(genann* neural_network, **double*** input_pointer, **double*** output_pointer, **double** expected_output, **double** learning_rate**)**

Train a single neuron, and immediately evaluate a predicted output using the neural network.

Parameter	Description
genann* neural_network	The pointer to the neural network
double* input_pointer	The pointer to an array of given input values
double* output_pointer	The pointer to an array of given output value(s)
double expected_output	A given, expected output value
double learning_rate	The learning rate of the neural network

Returns: A double value representing the error obtained by MAE.

---

void log_train_process(int epoch, **double** totalError, **FILE*** outputFile**)**

Log the error during a specific training epoch

Parameter	Description
int epoch	The current training epoch number
double totalError	The error value of the single training process
FILE* outputFile	The pointer to the output file (the error will be written in it)

Returns: This function does not return any value.

```
void train_once(genann* neural_network, double** inputs, double* outputs,
FILE* outputFile, int epoch, int training_iterations, double
learning_rate, int dataset_count, int log_error_interval)
```

The entry point to a single training iteration.

Parameter	Description
genann* neural_network	The pointer to the neural network
double** inputs	The pointer to an array of given input values
double* outputs	The pointer to an array of given output value(s)
FILE* outputFile	The pointer to the output file (the error will be written in it)
int epoch	The current training epoch number
int training_iterations	The total number of training iterations
double learning_rate	The learning rate of the neural network
int dataset_count	The total number of datasets
int log_error_interval	How often to log the error of the training process

Returns: This function does not return any value.

```
void train_neural_network(genann* neural_network, double** inputs, double*
outputs, FILE* outputFile, int training_iterations, double learning_rate,
int dataset_count, int log_error_interval)
```

The entry point to the entire training process. This calls the **train_once()** function repeatedly to train the neural network.

Parameter	Description
genann* neural_network	The pointer to the neural network
double** inputs	The pointer to an array of given input values
double* outputs	The pointer to an array of given output value(s)
FILE* outputFile	The pointer to the output file (the error will be written in it)
int training_iterations	The total number of training iterations
double learning_rate	The learning rate of the neural network
int dataset_count	The total number of datasets
int log_error_interval	How often to log the error of the training process

Returns: This function does not return any value.

Module: checksum.c

double evaluate_checksum(genann* neural_network**, double*** input**, int** check_input_size**)**

Evaluate the checksum of the input, using the trained neural network.

Parameter	Description
genann* neural_network	The pointer to the neural network
double* input	The pointer to an array of user inputs (for calculating checksum)
int check_input_size	The length of the input array

Returns: An evaluated checksum output.

---

double obtain_checksum(genann* neural_network**, int** max_test_input_length**)**

A wrapper function which requests user input, and obtains the checksum value.

Parameter	Description
genann* neural_network	The pointer to the neural network
int max_test_input_length	The maximum allowed input length

Returns: An evaluated checksum output.

Main Program: main.c

void release_memory(genann* neural_network**, double**** inputs**, double*** outputs**)**

Release the memory allocated for the neural network, the inputs array, and the outputs array.

Parameter	Description
genann* neural_network	The pointer to the neural network
double** inputs	The pointer to an array of given input values
double* outputs	The pointer to an array of given output value(s)

Returns: This function does not return any value.

References

GitHub repository of the genann library:

https://github.com/codeplea/genann.git

GitHub repository of this project:

https://github.com/yeahlowflicker/NCUDS-Neural-Network