The Optional Final Project

Developing an Artificial Neural Network Code from Scratch on C

(25 points based on the second marking scheme mentioned in course outline +**8** points bonus)

Pedram Pasandide

Due Date: 22 December 2023

Important Notes:

- This is an **optional** project. So **it is up to you** if you want to do it or not. In this course, we have two grading system. One with (Option 1) and the other one without a final project (Option 2) (Please refer to the course outline). If some students chose to do the final project (**Option 2**), then what we do is:

  1. Including the final project according to the marking scheme "with" final project (**Option 2**).
  2. Using the marking scheme "without" a final project (**Option 1**).

  And then we just use the better of the two grades! So you don't need to be worried about the final grade!

- This project is about Artificial Neural Network. If you produce good results, this will be one of the best projects that you are going to do during your undergraduate studies.

- For any reason if you have missed the lectures we had about ANN, I strongly suggest you to avoid this project!

1 Introduction

Download the latest version of `data.txt` and `main.c` from my GitHub. In this project you have to modify the ANN developed in `main.c` to train `data.txt`. Here is the structure of given data:

Based on the given data, we need a ANN model capable of predicting two binary outputs (y_1 and y_2) with given features (x_1 and x_2). Let's say if I give $x_1 = 0.961108775120079$ and $x_2 = 0.576862767986208$ to the model it should be able to predict $y_1^{predicted} = 0$ and $y_2^{predicted} = 1$, which are equal to the real values of y_1 and y_2 in the Table. So far what we know:

1. First layer (input layer l=1) has two inputs, including x_1 and x_2,

Table 1: Data in `data.txt`

Sample	x_1	x_2	y_1	y_2
1	0.961108775120079	0.576862767986208	0	1
2	0.891751713659224	0.619192234236843	0	1
3	0.955888707184407	0.729448225683374	0	1
...	...	...	...	...
48120	0.552926454894949	0.466761075682240	1	0

2. Last layer (output layer l=L) has two outputs (two neurons), including (y_1 and y_2),

3. Since the outputs are binary (either zero of one), we can use **Sigmoid()** activation function for both neurons in the last layer.

To answer the question that how many hidden layers we need to do the prediction, it is not always clear! As a rule to thumb, the more hidden layers and neurons model has, the more parameters (biases and weight) must be estimated, which can cause **over-fitting** in ANN! We must start with lower number of layers and neurons. If the accuracy for train data is not going up during the process of updating weights and bises (train) after a certain iteration (epoch), **one of the solutions** that might work is increasing the complexity of ANN (adding more hidden layers and neurons)! Here, based **on my experience on this specific data in the table**, I started with two hidden layers. First hidden layer (l=2) has 40 neurons($N^2 = 40$), and the second hidden layer (l=3) has 20 neurons ($N^3 = 20$). What we know so far:

4. The total number of layers in the ANN model is 4 ($L = 4$).

This is how the ANN model looks like:

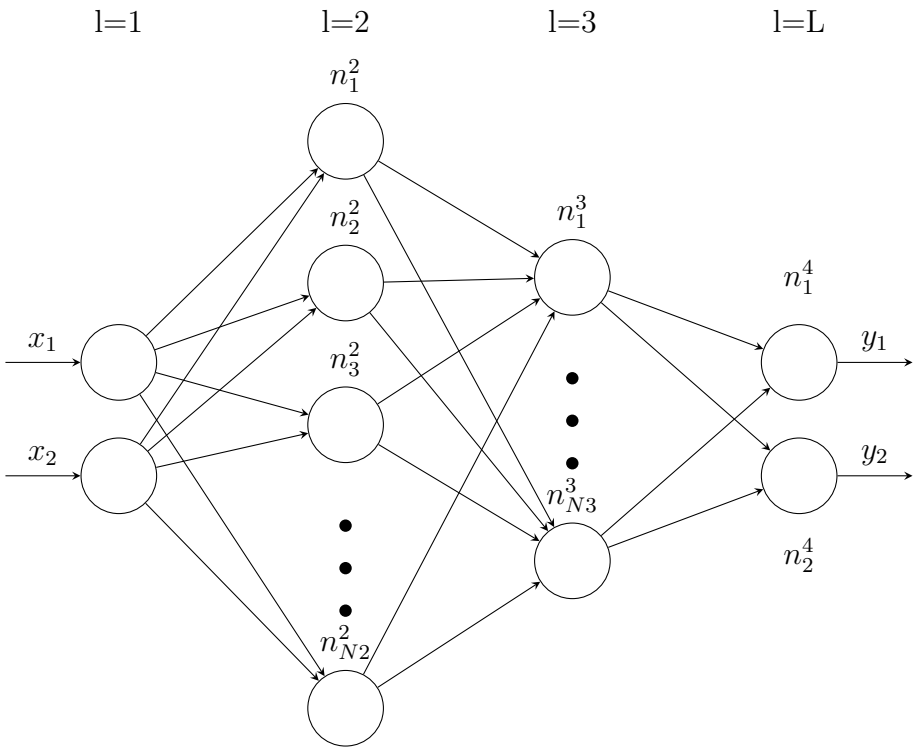
All the weights ans biases are randomly initialized within [-0.2, 0.2] using `<sodium.h>` library, defined by `initial_range` in the program. Probably you don't have the library installed, you need to install it first then during compile add `-lsodium` to link against the library. If you use `rand()` function you will not get the result!

A learning rate of 0.005 is set by `Learning_rate`. The number of iterations to do forward and backward propagation is set by `epochs`. `train_split` shows how many samples are used to train the model, let's say a value of `0.003` means 0.3% data will be used to train model. At this point if you have both `main.c` and `data.txt` files in the same directory you should be able to compile it using:

```
gcc -o ANN main.c -lsodium -lm
```

and running the program with:

```
./ANN
```



We discussed the printed results during the lectures. The results in every iteraions will be printed like:

```
epoch 32300:
Train Cost      0.047132,    Accuracy: 97.93%
Validation Cost 0.342040,    Accuracy: 77.82%
```

2 Improving the Code on C

2.1 Fixing the format of the code (10 points)

The code is like a mess! All the codes are in a single file, and many parts are hard-coded. Based on what we have learned during lectures, make a perfect structure for the code. Examples of what you can do is:

- Currently the process of reading `data.txt` is happening inside the `main.c`. `main.c` is where you only call the functions and the real implementation must be before `int main ()`, the same as functions `random_double`, `sigmoid`, and `ForwardPass()`. Make a function to read the data from `.txt` file, call the function in the `int main()` and save the values in a array named `double data[MAX_ROWS][MAX_COLS]`.

- Move the backward propagation implementation, which is currently in `int main()`, to a function named `BackwardPass()`, the same as `ForwardPass()`.
- Currently the algorithm is to:
  1. Read the data.
  2. Initialize randomly weights and biases
  3. For each iteration in `for (int ep = 0; ep < epochs; ep++)`:
     (a) Call forward and backward propagation to update the weights.
     (b) And every 100 times (`ep%100==0`) print out accuracy and cost function for both validation and train data set.

  Make a new function called `Evaluation()`, which will hold the steps 2 and 3 (both (a) and (b)). In this case, the only function will be called in `int main()` are reading the data and `Evaluation()`.
- In the last step, split you code into multiple files the same as the second assignment. The `main.c` file contains only `int main()`. The declaration of all other functions must be in a file called `mymodel.h` and the real implementation of there functions must be in `mymodel.c`. You can keep the constant values defined by `#define`

2.2 Dynamic Memory Allocation (5 points)

When the `train_split` (TS) is equal to 0.1 you will get `Segmentation fault (core dumped)`. Why? to find the answer, you need to use `gdb` or `lldb` to find which line of the code this is happening?

This is due the fact that stack doesn't have enough memory to keep all the weights for all the train data set. You must use dynamic memory allocation for all the variable. Send them to functions as pointer. In this part of you have to deal with the memory and make sure the code is optimized. This part is going to be difficult if you do not understand the codes I gave you.

2.3 Inputs Given by the User (2 points)

This part of the program is hard coded, and when we want to conduct analysis for different inputs, we have to recompile the source code again.

```
#define num_inputs 2          // N1 = 2
#define num_neurons_layer2 40 // N2 = 40
#define num_neurons_layer3 20 // N3 = 20
```

```
#define num_outputs 2          // N4 = 2
#define Learning_rate 0.005    // 0 - 1, 0.01 - 0.0001
#define epochs 100000
#define train_split 0.003      // 0.3 percent of data will be used for train
```

Change the code format to receive these values from the user. At this point I have to be able to run your program with:

```
./ANN <epochs> <Learning_rate> <train_split> <num_inputs> <num_neurons_layer2> <
    num_neurons_layer3> <num_outputs>
```

For example:

```
./ANN 100000 0.005 0.003 2 40 20 2
```

where `2 40 20 2` are the number of neurons in the layer 1, 2, 3 and 4, respectively. The layers of 2 and 3 are called hidden layers.

2.4 Increasing the Accuracy and Avoiding Over-fitting (8 points)

Over-fitting usually occurs when there is a huge gap between the accuracy of the train and validation data. To avoid it, training starts with lower number of hidden layers and neurons. The other ways is to pick a lower number of epochs when the accuracy for validation is not going higher anymore. At this moment the accuracy of the model is around 80%. Change the parameters mentioned above, and increase or decrease the number of neurons in the layers two and three ($l = 2$ and $l = 3$) such that the accuracy of validation data set is higher than 97%, and there is no sign of over-fitting!

The most important way so see how you can improve your model during training is to carrying out some **Sensitivity Analysis** to see the effects of different parameters on the accuracy. Produce some results by changing values of `Learning_rate`, `epochs`, and `train_split`. and provide the following format of tables (Table 2, Table 3, Table 4) in your `ReadME.tex` file.

Table 2: epochs = 100000, train_split = 0.003, **Learning Rate (LR) is variable**

LR	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
0.0005				
0.001				
0.005				
0.01				
0.1				

A. Give me detail description here about these results.

Table 3: Learning_rate = 0.005, train_split = 0.003, **epochs is variable**

epochs	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
100				
1000				
10000				
100000				

Table 4: epochs = 100000, Learning_rate = 0.005, **train_split (TS) is variable**

TS	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
0.0003				
0.003				
0.005				
0.01				
0.1				
0.2				
0.4				
0.6				
0.8				

B. I have uploaded the code in Python (`ANN.ipynb`) on my GitHub and we did the same thing on Python by using ANN library like `tensorflow`. You can run the code on Google Colab if you don't have Python on your OS. Which one is better to work with? On C or Python?

2.5 Changing the shape of ANN (+8 bonus points)

At this moment we have 2 number of hidden layer which are hard-coded. But what if we want to have 1 OR 3 Or other number of hidden layers? At this moment, based on `data.txt`, the number of inputs and outputs both are 2. What if I want to use the model on another dataset which has different number of inputs and outputs? This is why the ANN we have so far is completely useless!

Warning: This part is extremely difficult, and I don't suggest you to waste your time on this if you are busy with exams! But if you do it, you will have a really precious code!

2.5.1 Changing the Number of Hidden Layers (4 points/8 bonus)

So far you should be able to run your code with the following command:

```
./ANN <epochs> <Learning_rate> <train_split> <num_inputs> <num_neurons_layer2> <
    num_neurons_layer3> <num\outputs>
```

Change the code to accept the number hidden layers from user. For example if we have 5 hidden layers, then:

```
./ANN <epochs> <Learning_rate> <train_split> <num_inputs> <num_neurons_layer2> <
    num_neurons_layer3> <num_neurons_layer4> <num_neurons_layer5>  <num_neurons_layer6> <
    num_outputs>
```

For example:

```
./ANN 100000 0.005 0.003 2 40 20 60 70 10 2
```

where `40 20 60 70 10`, are the number of neurons in the hidden layers.

If I want to have 3 hidden layers, then the input format is:

```
./ANN <epochs> <Learning_rate> <train_split> <num_inputs> <num_neurons_layer2> <
    num_neurons_layer3> <num_neurons_layer4>  <num_outputs>
```

For example:

```
./ANN 100000 0.005 0.003 2 40 20 60 2
```

where `40 20 60`, are the number of neurons in the hidden layers.

2.5.2 Changing the Number Inputs and Outputs (4 points/8 bonus)

To test this part you need another dataset, with the same format of `data.txt`, but with different number of columns. In another word, a dataset that has different number of inputs (x) and different number of outputs (y). `data.txt` has 2 inputs and 2 outputs, that's why the first and the last layers of the ANN need 2 and 2 neurons, respectively. So far in all the run the input values of `num_inputs` `num_outputs` were equal to 2. After doing this part, I have to be able to run you code using different datasets!

3 Report and `Makefile`

Provide a `Makefile` to compile your code, and in the report with details tells us how your `Makefile` works. Your report **must** be in `README.tex` file. Other formats will not be accepted. Create the `README.pdf` file from `README.tex`

4 Submission

Submit On Avenue to Learn following files:

1. `main.c`
2. `mymodel.c`
3. `mymodel.h`
4. `Makefile`
5. `README.pdf` and `README.tex`, a detailed and perfect description!

5 Future Directions

- At this point we have still only one activation function. The choice of activation functions depends on the dataset we have. So it must be an option that user picks.
- We do not save the model! We train the model, then we need to save the weights (lets say in a `.txt` or EXCEL file) and have a function to read these weight and use them for predictions on new data.
- There are many more future directions, but I am tired to keep writing! You can always contact me if you have any questions. I am new in this field of study, and my knowledge is very limited in this field, but I try to answer your questions.