Project: Artificial Neural Network

with 7% bonus of total grade

Pedram Pasandide

Due Date: 13 August

For any reason if you have missed the past two lectures, I suggest you to do the other project! 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:

Table 1: Data in `data.txt`

samples	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

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:

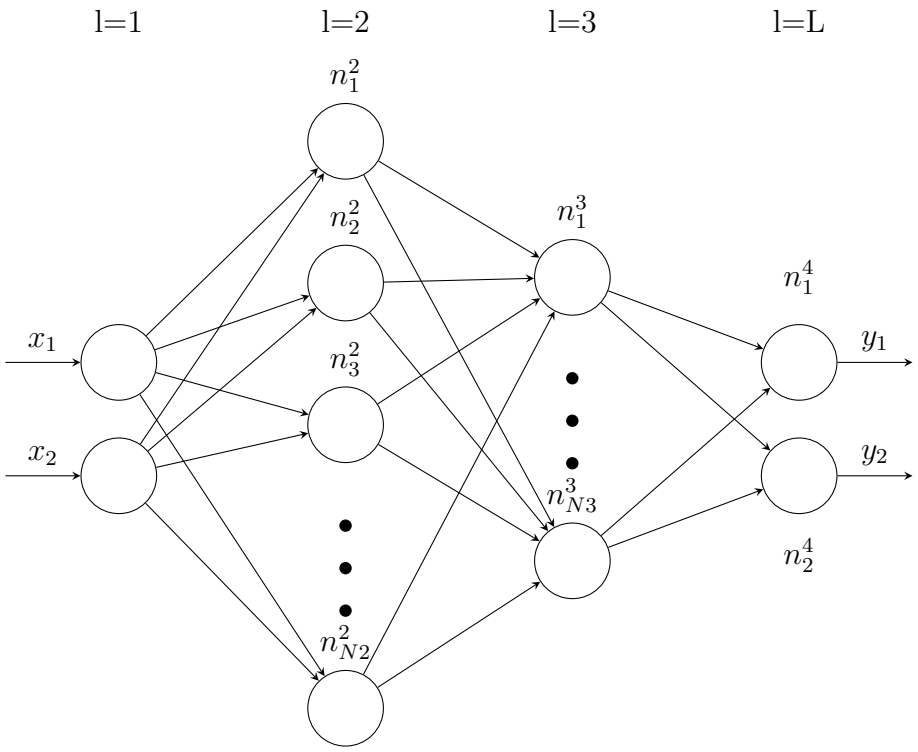
1. First layer (input layer l=1) has two inputs, including x_1 and x_2,

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. 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.

Part A. (**Total 30 points**) The code is like a mess! Based on what we have learned during lectures, make a perfect structure for the code. Examples of what you can do is:

- (**5/25 points**) Currently the process of reading `data.txt` is happening inside the `main.c`. `main.c` is where you only call you 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]`.
- (**8/25 points**) Move the backward propagation implementation, which is currently in `int main()`, to a function named `BackwardPass()`, the same as `ForwardPass()` function.
- (**7/25 points**) 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++)` do the forward and backward propagation to update the weights. And every 100 times (`ep%100==0`) print out some accuracy and cost function for both validation and train data set.

Make a new function called `Evaluation()`, which gives us the accuracy and cost values for both validation and train data set. So, this part of the code is also going to be transferred into a function instead of keeping the implementation inside `int main`.

- (**5/25 points**) In the last step, split you code into multiple files the same as second assignment. The `main.c` file contains only `int main`. Write constant values defined by `#define` must be in a file named `constants.h` with proper comments to describe the purpose of the constant. Let's say, `#define Learning_rate 0.005` is the learning rate in backward propagation. 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`.

Part B. (**Total 15 points**) 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** (**3/15 points**)

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

Table 3: Learning_rate = 0.005, train_split = 0.003, **epochs is variable** (**3/15 points**)

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

Give me detail description here about these results (**6/15 points**).

- The advantages and disadvantages of decreasing and increasing these parameters. You can use ChatGPT and only if you understand it you can include them here.

Table 4: epochs = 100000, Learning_rate = 0.005, **train_split (TS) is variable (3/15 points)**

Test	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
TS = 0.0003				
TS = 0.003				
TS = 0.005				
TS = 0.01				
TS = 0.1				

- Also 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?

Part C. (**+5 bonus points**) **Over-fitting usually** occurs when there is a huge gap between the accuracy of train and validation. To avoid it, training starts with lower number of hidden layers and neurons. The other ways is to pick lower number of epochs when the accuracy for validation is not going higher. 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!

Part D. (**+2 bonus points**) I have uploaded the code in python on my GitHub but by using ANN library like `tensorflow`. Which one is better to work with in terms of ANN models? C of Python?

Submit On Avenue to Learn following files:

1. `main.c`
2. `constants.h`
3. `mymodel.c`
4. `mymodel.h`
5. `Makefile`
6. `ReadME.tex` Make sure this is a LaTeX format file. Descriptions in other formats will not be accepted.