Final Grade Prediction Based On Quiz Average and Final Exams Using Fast Artificial Neural Networks (FANN) (November 2015)

Marvin Jay O. Limjuco, and Dex Miguel F. Samson., *Students, DLSU-Manila*

Abstract—**Final grade prediction has always been an integral part of LaSallian Engineering culture. Fast Artificial Neural Networks can be used to give students an insight on the most probable grade they can achieve given a set of data composed of their quiz average and final exam mark. The method by which the predicted grade is calculated is accomplished by training the Artificial Neural Network with data gathered from previous quiz averages and final exam marks of students together with the final grade they received for the specific subject. The goal of the project is to achieve at least an 80% accuracy in predicting the final grade of the student. This project encompasses the lessons and theories that were discussed in Machine Intelligence and provides an avenue for the students to showcase what they have learned throughout the term.**

I. Introduction

Balancing academics and proper prioritization are necessities of students in the University. Through the use of Fast Artificial Neural Networks, students are given an efficient way of knowing the probable grade they are to receive in a subject given their quiz average and how well they perform in their final exam. By having an idea on what grade they would most probably get, students can prioritize subjects that need higher final exam marks and can therefore lessen their chances of failing subjects during the term. This, in turn, can help keep their academic flowchart in-tact and effectively reduce their chances of having back-subjects which could prolong their stay in the University. The benefits of knowing which subjects to properly prioritize may also reduce the financial burden incurred by their parents – thereby allowing for money to be invested in other important matters such as health and insurance, among others.

II. Dataset

The dataset to be used in the project was gathered from Lasallian Computer Engineering students who primarily enrolled in the year 2010. Their quiz averages, final exam marks, and final grades in their major subjects: DIGITDE, CSARCHI, and MPROSYS were compiled and are used as the training data for the project. Plotting these data resulted in a scattered plot that cannot be realized by simple linear regression. The use of Artificial Neural Networks was found to be the most effective means in understanding this data.

III. Artificial Neural Network

A neural network is a network made up of three layers: the input layer, the hidden layer, and the output layer. However, the hidden layer may be made up of several layers, depending on the number of neurons being used in a certain function. In contrast, an artificial neural network is used to represent non-linear hypotheses and when there are numerous sample data. The Fast Artificial Neural Network (FANN) library is used in this project. Additionally, the FANN library provides several functions that essentially handle the given neural network.

There are usually two modes in which a neural network operates: training and execution. During training, the data set is used to calculate the weights of each neuron before forward propagating it to the next layer. In FANN, the function "fann_train_rprop" is responsible for performing these operations. It is an adaptive function which means it does not have a standard learning rate. In this project, the function will be used in order to get the most probable output. The two inputs, the quiz average and finals score, is used to train and validate the most probable grade. In the project, 70 out of 110 of the data will be allotted for training, 20 will be allotted for validation, and the remaining 20 will be allotted for testing. Testing works hand-in-hand with training and where the mean squared error (MSE) is computed.

IV. Model Representation and Calculations

$$h_\theta(x) = \frac{1}{(1+e^{-z})} \qquad \text{eq. 1}$$

$$a_1^{(2)} = g(\theta X_0 + \theta X_1 + \theta X_2) \qquad \text{eq. 2}$$

The neurons contained in the hidden layers uses the sigmoid function given in Equation 1. In this project, a single neuron in the second layer is basically a sum of the weighted activation or sigmoid functions from the previous layer. This source code demonstrates how the weighted neurons from layer 1 is computed in layer 2:

```
        //a neuron in layer 2 having a distinct
weight
        Layer2In[j] = Weight[0][j] ;
        for( i = 1 ; i <= NumUnits1 ; i++ ) {
                //summation of the weighted neurons
        from layer 1
```

```
            Layer2In[j]   +=   Layer1Out[i]   *
        Weight[i][j];
        }
        //each neuron in layer 2 outputs a sigmoid
function
        Layer2Out[j]    =    1.0/(1.0    +    exp(-
Layer2In[j])) ;
```

Figure 1 shows the representation of how the artificial neural network and equation 2 shows the activation function of a certain neuron in the second layer.

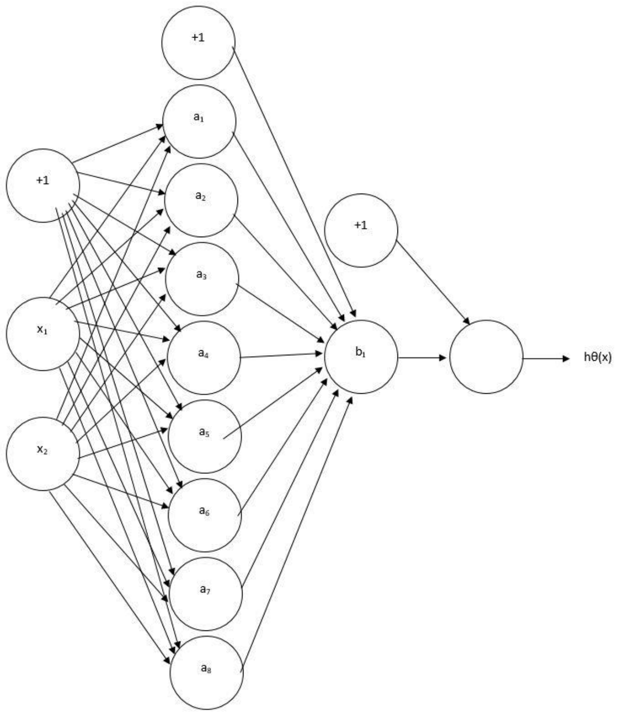

Fig. 1 – Representation of the Artificial Neural Network

V. Cost Function Plot

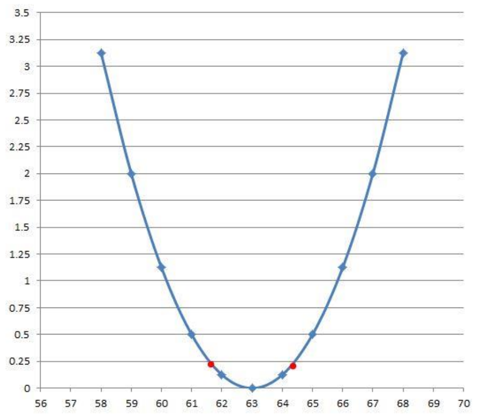

Fig. 2 - CSARCHI Cost Function of 1.0

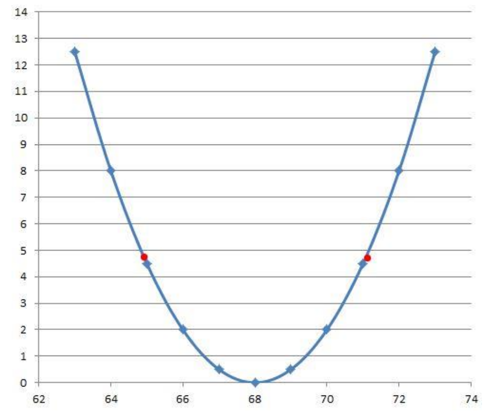

Fig. 3 - CSARCHI Cost Function of 1.5

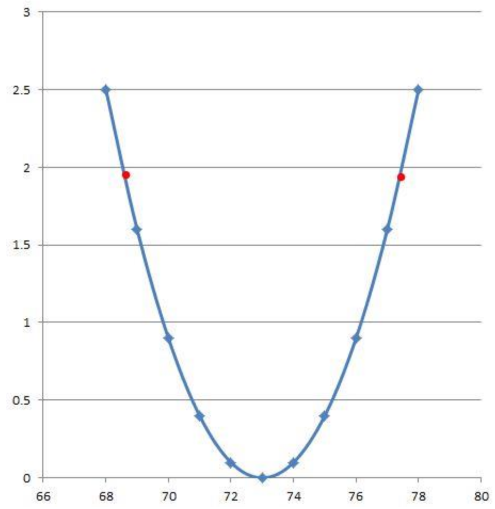

Fig. 4 - CSARCHI Cost Function of 2.0

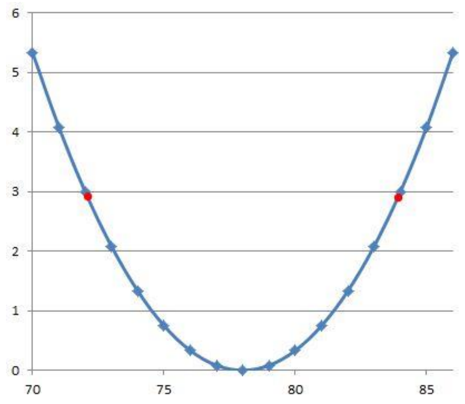

Fig. 5 - CSARCHI Cost Function of 2.5

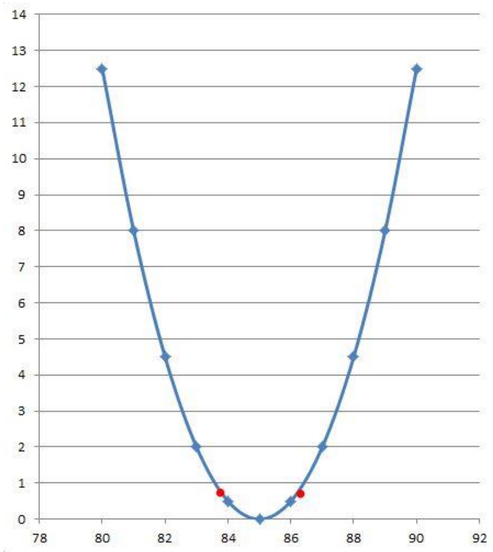
Fig. 6 - CSARCHI Cost Function of 3.0

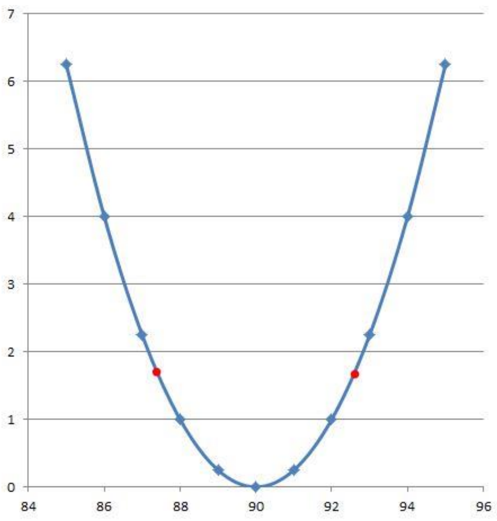
Fig. 7 - CSARCHI Cost Function of 3.5

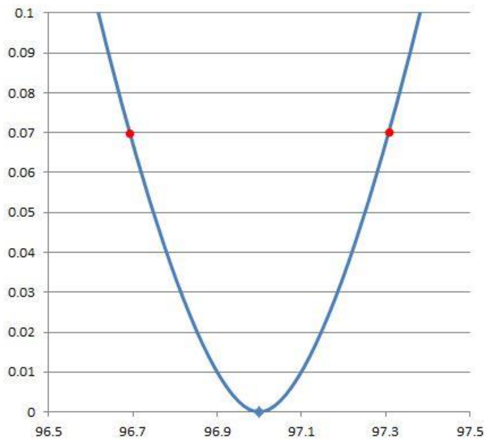
Fig. 8 - CSARCHI Cost Function of 4.0

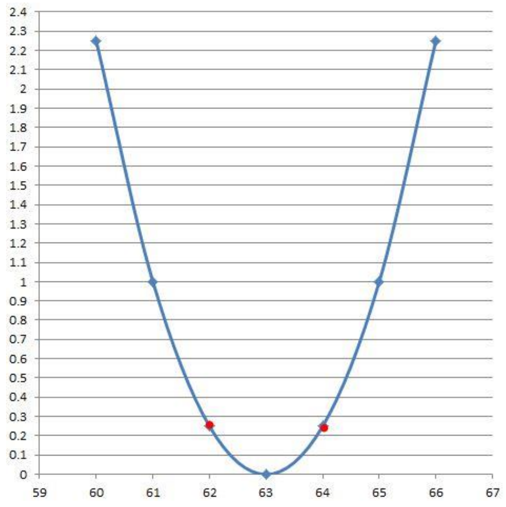
Fig. 9 - DIGITDE Cost Function of 1.0

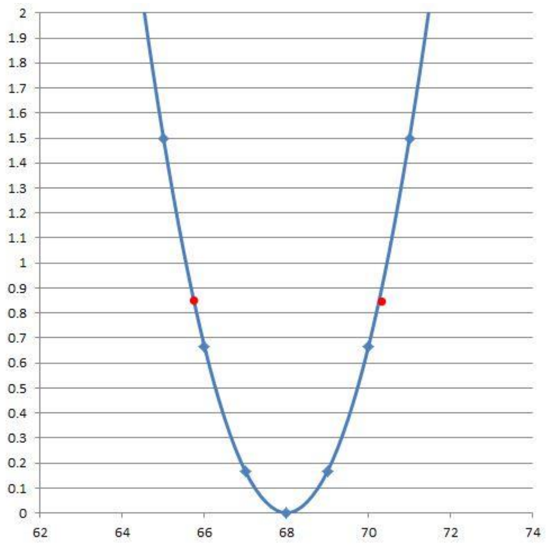
Fig. 10 - DIGITDE Cost Function of 1.5

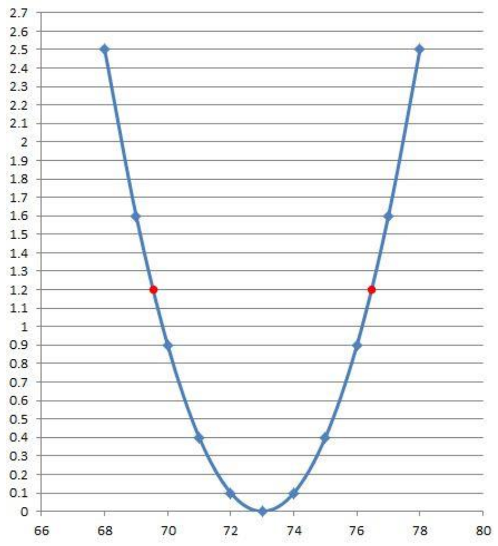
Fig. 11 - DIGITDE Cost Function of 2.0

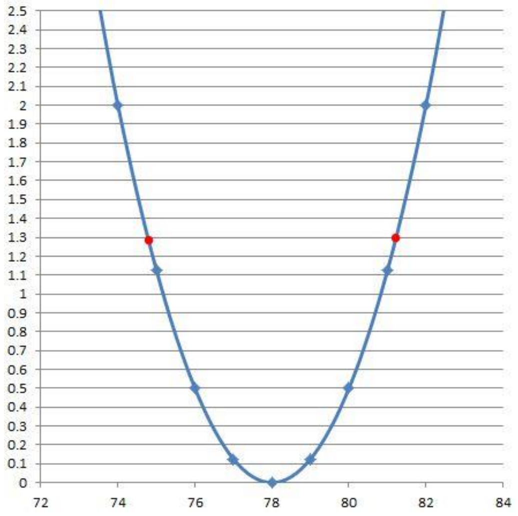
Fig. 12 - DIGITDE Cost Function of 2.5

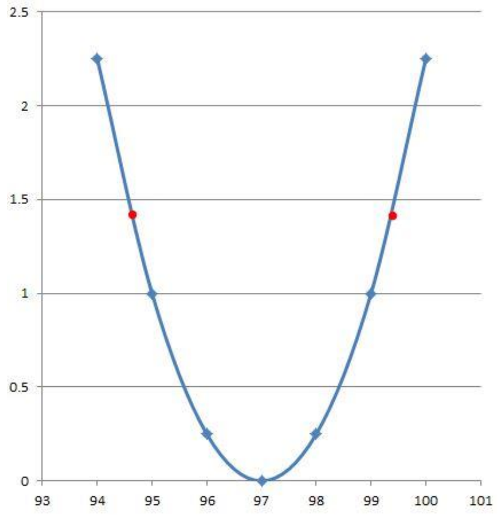
Fig. 15 - DIGITDE Cost Function of 4.0

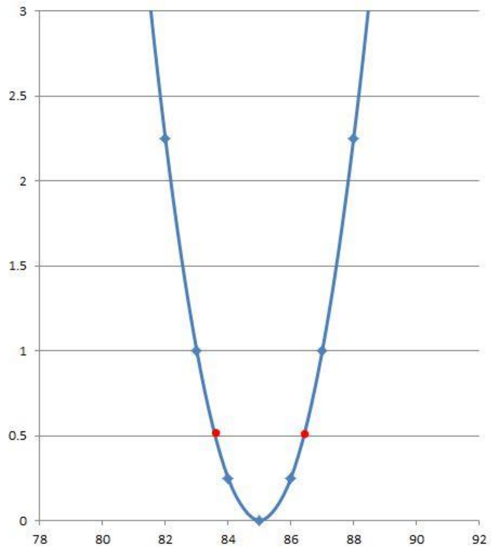
Fig. 13 - DIGITDE Cost Function of 3.0

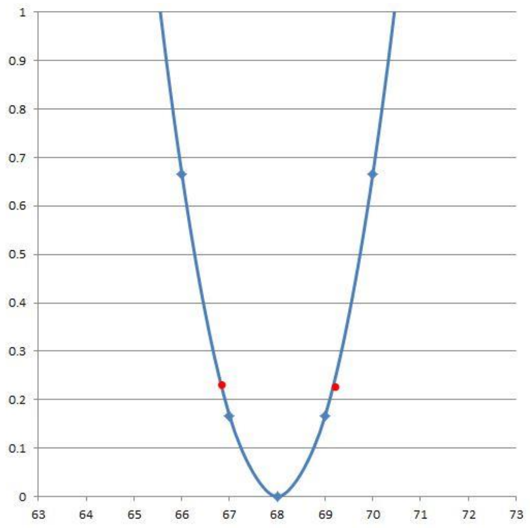
Fig. 16 - MPROSYS Cost Function of 1.5

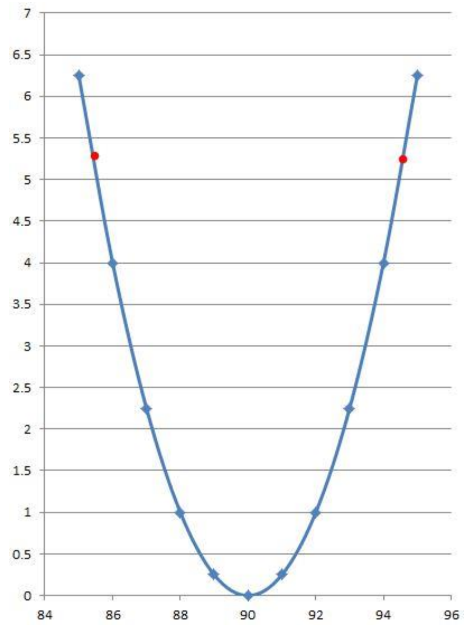
Fig. 14 - DIGITDE Cost Function of 3.5

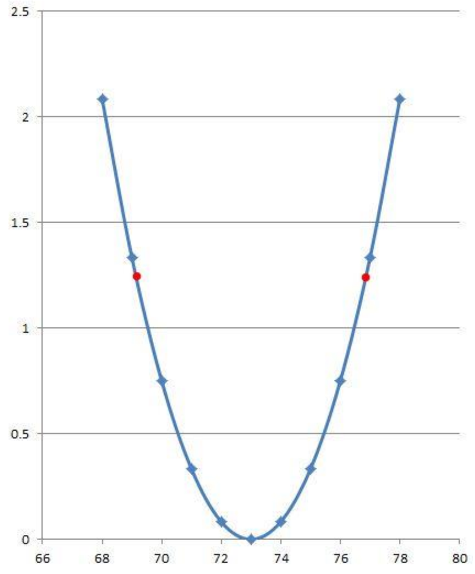
Fig. 17 - MPROSYS Cost Function of 2.0

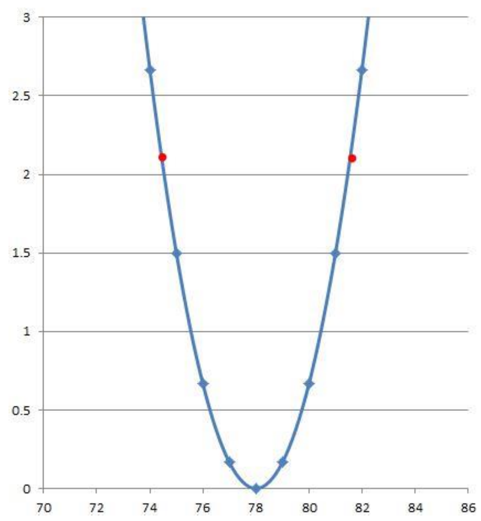

Fig. 18 - MPROSYS Cost Function of 2.5

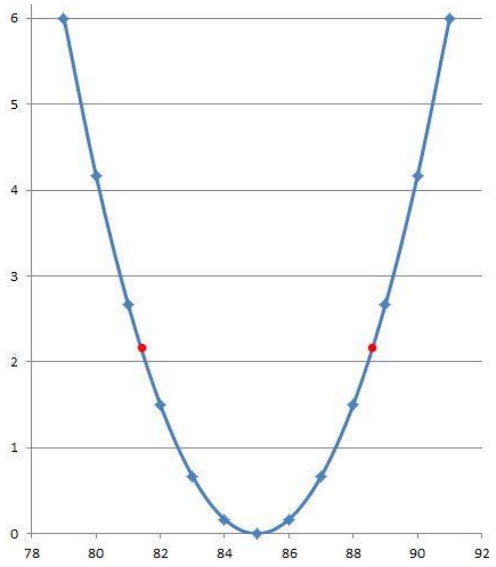

Fig. 19 - MPROSYS Cost Function of 3.0

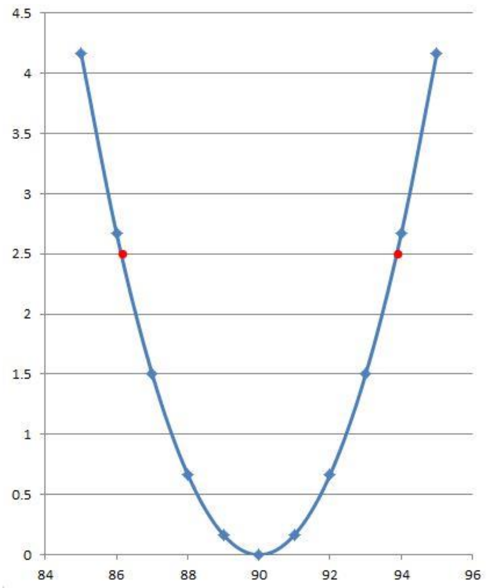

Fig. 20 - MPROSYS Cost Function of 3.5

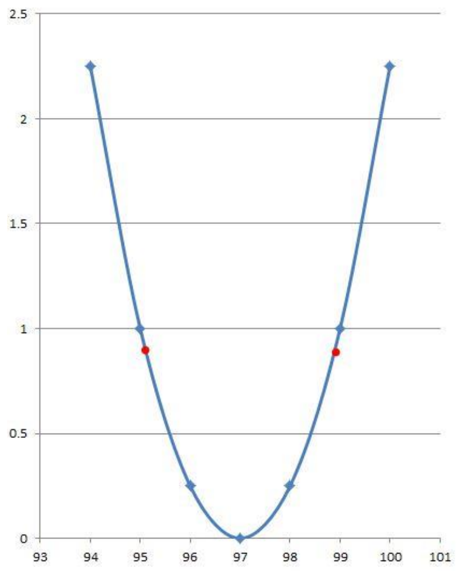

Fig. 21 - MPROSYS Cost Function of 4.0

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2\right) \qquad \text{eq. 3}$$

Figures 2 to 21 shows the cost function for each grade in each subject. This function is represented in equation 3. It basically shows the difference of the generated output from the real/actual output. For example, in figure 20, it shows the cost function plot for 3.5. If the generated output produced a grade of 90, then the cost is at minimum. Additionally, a certain mark is essentially made up of several possible grades. With that being said, there is a region where in a certain cost is still accepted, as long as it is inside the grade range.

VI. Mean – Squared Error Plots

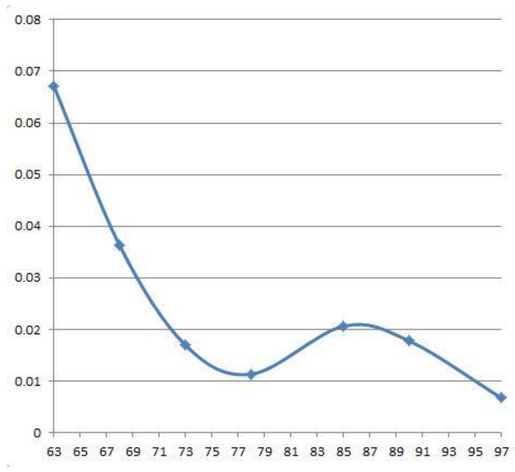

Fig. 22 - MSE Plot of CSARCHI

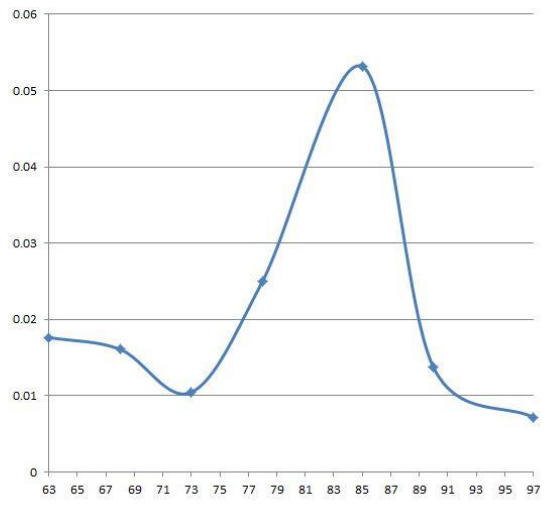

Fig. 23 - MSE Plot of DIGITDE

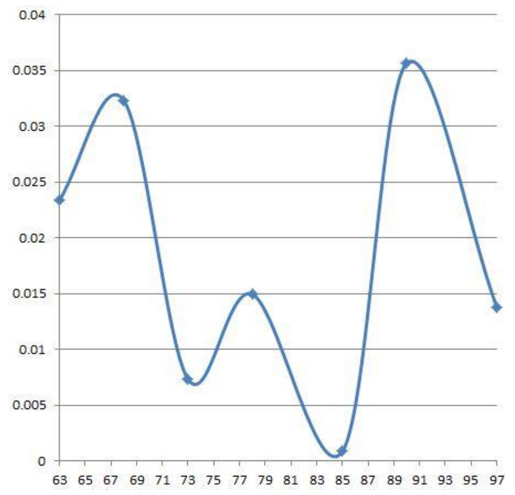

Fig. 24 - MSE Plot of MPROSYS

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2 \qquad \text{eq. 4}$$

The mean squared error was computed after the final testing. Equation 4 shows the formula to compute the MSE. This equation basically shows the squared difference of the expected output from the actual output. The main reason why each of the MSE plots shows a unique curve is because of the data set it follows. Grades vary from each subject which affects how the FANN essentially produces it weights. For example, in figure 23, the x-axis ranged from 63 to 97, which shows students who took this class, got at least a 2.5. In other subjects such as MPROSYS, as shown in figure 24, the x-axis also ranged from 63 to 97. Meaning, students who took this class got final grades ranging from 1.0 to 4.0. Additionally, an MSE plot shows how the neural network is based. Looking at figure 22, the curve is at minimum when it reaches 78. This means that most of the grades in the data set got a 2.5.

VII. Confusion Matrix

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	2	6						8
1.5	1	3	5					9
2.0		3	14	2				19
2.5		1	5	7	1			14
3.0			1	3	7			11
3.5						3		3
4.0						3	3	6
								70

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	🟩	1						1
1.5		🟩	2					2
2.0			7	1				8
2.5				2				2
3.0					2			2
3.5						3		3
4.0							2	2
								20

Fig. 25 - Train and Test Confusion Matrix of CSARCHI

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	6	1						7
1.5	1	5	2					8
2.0		1	18	1				20
2.5			1	10				11
3.0			1	2	5			8
3.5						10		10
4.0						1	5	6
								70

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	1							1
1.5		3	1					4
2.0			4	1				5
2.5			1	3				4
3.0				1	1			2
3.5						3		3
4.0							1	1
								20

Fig. 26 - Train and Test Confusion Matrix of DIGITDE

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	5	3						8
1.5	1	3	6					10
2.0		4	13	2				19
2.5			3	6				9
3.0			1	2	6			9
3.5					1	7		8
4.0						2	5	7
								70

	1.0	1.5	2.0	2.5	3.0	3.5	4.0	total
1.0	3							3
1.5		1	2					3
2.0			5					5
2.5				4				4
3.0					1			1
3.5					1	2		3
4.0							1	1
								20

Fig. 27 - Train and Test Confusion Matrix of MPROSYS

Figures 25 to 27 represent a confusion matrix after training and testing the data set. This matrix is used to describe the performance of the training and testing process where the values are classified as to whether it achieved the correct data. For example, in figure 27, it is shown in the first row that the training process generated three correct 1.0 grades out of the possible three. Consequently, the second row only produced one correct 1.5 grades out of the possible three.

VIII. ACCURACY

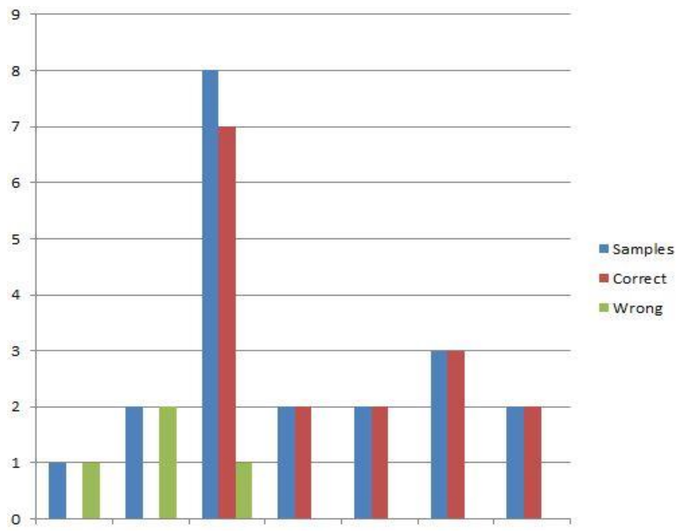

Fig. 25 - Accuracy Plot of CSARCHI

ACCURACY: 16/20 = 80%

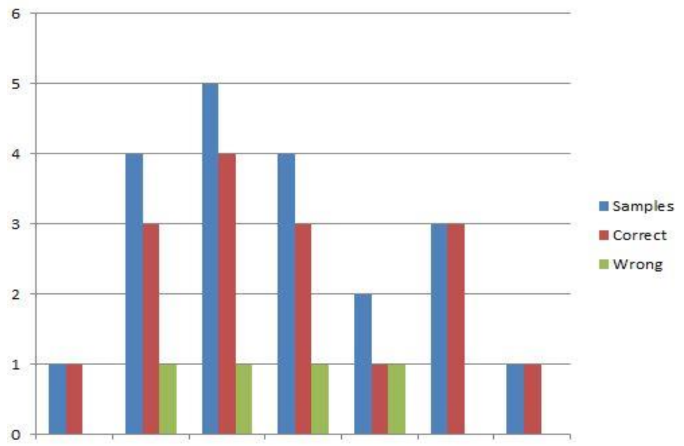

Fig. 26 - Accuracy Plot of DIGITDE

ACCURACY: 16/20 = 80%

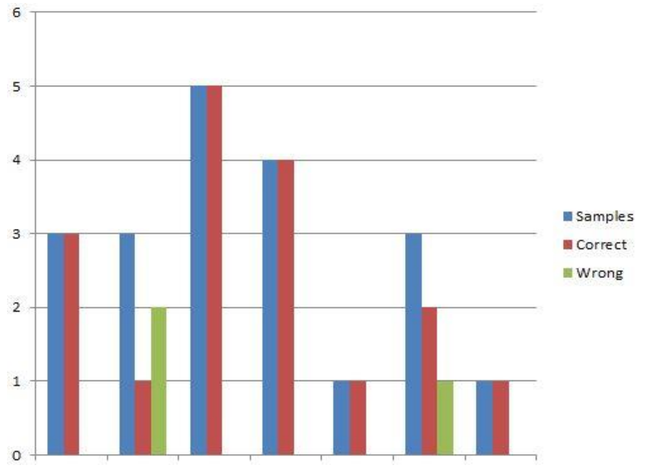

Fig. 27 - Accuracy Plot of MPROSYS

ACCURACY: 17/20 = 85%

$$Accuracy = 1 - \frac{|Actual - Experimental|}{Actual} \quad \text{eq. 5}$$

Figures 25 to 27 represent the accuracy of the neural network and how well it produces the correct grade. The values were based after the testing process. Collectively, each subject produced an average around 83% accuracy, which is one of the objectives of this project. Each plot represents the accuracy of the generated values of the testing process compared to the actual data. For example, in figure 27, the number of samples for 1.0 is three; it shows that the neural network used also produced three correct outputs. On the other hand, 1.5 only got two of the possible three correct outputs. The formula for accuracy is shown is equation 5.

IX. C CODES

train.c

```
int FANN_API test_callback(struct fann *ann, struct
fann_train_data *train,
  unsigned      int      max_epochs,      unsigned      int
epochs_between_reports,
  float desired_error, unsigned int epochs)
{
  printf("Epochs          %8d.  MSE:  %.5f.  Desired-MSE:
%.5f\n", epochs, fann_get_MSE(ann), desired_error);
  return 0;
}

int main()
{
  fann_type *calc_out;
  const unsigned int num_input = 2;
  const unsigned int num_output = 1;
  const unsigned int num_layers = 3;
  const unsigned int num_neurons_hidden = 12;
  const float desired_error = (const float) 0;
  const unsigned int max_epochs = 100000;
  const unsigned int epochs_between_reports = 1000;
  struct fann *ann;
  struct fann_train_data *data;

  unsigned int i = 0;
  unsigned int decimal_point;

  printf("Creating network.\n");
  ann   =   fann_create_standard(num_layers,   num_input,
num_neurons_hidden, num_output);

  data                                                  =
fann_read_train_from_file("csarchi_train.data");

  fann_set_activation_steepness_hidden(ann, 0.5);
  fann_set_activation_steepness_output(ann, 0.5);

  fann_set_activation_function_hidden(ann,
FANN_SIGMOID);
  fann_set_activation_function_output(ann,
FANN_SIGMOID);

  fann_set_train_stop_function(ann, FANN_STOPFUNC_BIT);
  fann_set_bit_fail_limit(ann, 0.01f);

  fann_set_training_algorithm(ann, FANN_TRAIN_RPROP);
  fann_init_weights(ann, data);

  printf("Training network.\n");
  fann_train_on_data(ann, data, max_epochs,
epochs_between_reports, desired_error);

  printf("Testing network. %f\n", fann_test_data(ann,
data));

  for(i = 0; i < fann_length_train_data(data); i++)
  {
    calc_out = fann_run(ann, data->input[i]);
    printf("GG test (%f,%f) -> %f, should be %f,
difference=%f\n",
          data->input[i][0], data->input[i][1],
calc_out[0], data->output[i][0],
          fann_abs(calc_out[0] - data->output[i][0]));
  }

  printf("Saving network.\n");

  fann_save(ann, "csarchi_train_float.net");

  decimal_point = fann_save_to_fixed(ann,
"csarchi_train_fixed.net");
  fann_save_train_to_fixed(data,
"csarchi_train_fixed.data", decimal_point);

  printf("Cleaning up.\n");
  fann_destroy_train(data);
  fann_destroy(ann);

  return 0;
}
```

validation.c

```
int FANN_API test_callback(struct fann *ann, struct
fann_train_data *train,
  unsigned int max_epochs, unsigned int
epochs_between_reports,
  float desired_error, unsigned int epochs)
{
  printf("Epochs %8d. MSE: %.5f. Desired-MSE:
%.5f\n", epochs, fann_get_MSE(ann), desired_error);
  return 0;
}

int main()
{
  fann_type *calc_out;
  const unsigned int num_input = 2;
  const unsigned int num_output = 1;
  const unsigned int num_layers = 3;
  const unsigned int num_neurons_hidden = 12;
  const float desired_error = (const float) 0;
  const unsigned int max_epochs = 100000;
  const unsigned int epochs_between_reports = 1000;
  struct fann *ann;
  struct fann_train_data *data;

  unsigned int i = 0;
  unsigned int decimal_point;

  printf("Creating network.\n");
  ann = fann_create_standard(num_layers, num_input,
num_neurons_hidden, num_output);

  data =
fann_read_train_from_file("csarchi_validation.data");

  fann_set_activation_steepness_hidden(ann, 0.5);
  fann_set_activation_steepness_output(ann, 0.5);

  fann_set_activation_function_hidden(ann,
FANN_SIGMOID);
  fann_set_activation_function_output(ann,
FANN_SIGMOID);

  fann_set_train_stop_function(ann, FANN_STOPFUNC_BIT);
  fann_set_bit_fail_limit(ann, 0.01f);

  fann_set_training_algorithm(ann, FANN_TRAIN_RPROP);

  fann_init_weights(ann, data);

  printf("Training network.\n");
  fann_train_on_data(ann, data, max_epochs,
epochs_between_reports, desired_error);

  printf("Testing network. %f\n", fann_test_data(ann,
data));

  for(i = 0; i < fann_length_train_data(data); i++)
  {
    calc_out = fann_run(ann, data->input[i]);
    printf("GG test (%f,%f) -> %f, should be %f,
difference=%f\n",
          data->input[i][0], data->input[i][1],
calc_out[0], data->output[i][0],
          fann_abs(calc_out[0] - data->output[i][0]));
  }

  printf("Saving network.\n");

  fann_save(ann, "csarchi_validation_float.net");

  decimal_point = fann_save_to_fixed(ann,
"csarchi_validation_fixed.net");
  fann_save_train_to_fixed(data,
"csarchi_validation_fixed.data", decimal_point);

  printf("Cleaning up.\n");
  fann_destroy_train(data);
  fann_destroy(ann);

  return 0;
}
```

test.c

```
int main()
{
  fann_type *calc_out;
  unsigned int i;
  int ret = 0;

  struct fann *ann;
  struct fann_train_data *data;

  printf("Creating network.\n");

#ifdef FIXEDFANN
  ann =
fann_create_from_file("csarchi_train_fixed.net");
#else
  ann =
fann_create_from_file("csarchi_train_float.net");
#endif

  if(!ann)
  {
    printf("Error creating ann --- ABORTING.\n");
    return -1;
  }

  fann_print_connections(ann);
  fann_print_parameters(ann);

  printf("Testing network.\n");

#ifdef FIXEDFANN
  data                                            =
fann_read_train_from_file("csarchi_test_fixed.data");
#else
  data = fann_read_train_from_file("csarchi_test.data");
#endif

  for(i = 0; i < fann_length_train_data(data); i++)
  {
    fann_reset_MSE(ann);
    calc_out  =  fann_test(ann,  data->input[i],  data-
>output[i]);
#ifdef FIXEDFANN
    printf("GG  test  (%d,  %d)  ->  %d,  should  be  %d,
difference=%f\n",
         data->input[i][0],           data->input[i][1],
calc_out[0], data->output[i][0],
         (float)    fann_abs(calc_out[0]    -    data-
>output[i][0]) / fann_get_multiplier(ann));

    if((float)     fann_abs(calc_out[0]     -     data-
>output[i][0]) / fann_get_multiplier(ann) > 0.2)
    {
      printf("Test failed\n");
      ret = -1;
    }
#else
    printf("GG  test  (%f,  %f)  ->  %f,  should  be  %f,
difference=%f\n",
         data->input[i][0],           data->input[i][1],
calc_out[0], data->output[i][0],
         (float)    fann_abs(calc_out[0]    -    data-
>output[i][0]));
#endif
  }

  printf("Cleaning up.\n");
  fann_destroy_train(data);
  fann_destroy(ann);

  return ret;
}
```

X. CONCLUSION

The project aims to provide an effective means of predicting the final grade of a subject based on quiz average and the final exam mark. The Artificial Neural Network created was able to sustain an accuracy greater than 80% throughout the validation and testing process which the group deems accurate. This was achieved by having an iteration of 100k and a steepness value of 0.5 which is suited for function approximation. All in all, the project achieved its purpose and goal of effectively predicting the final grade of a student in a certain subject.