Tuition & Fee Prediction

Aquino, Patrica Rose
Carag, Stephanie
Jacinto, Dan Emanuel
Lunasco, Jan Osbert

***Abstract*—This project is about the application of Artificial Neural Network on tuition and fees prediction for specifically for De La Salle Universitys degree program Bachelor of Science in Computer Engineering (BS-CPE). The data came from different Enrollment Assessment Forms (EAFs) of students which shows the assessment of the students the total tuition and fees for a particular term. Using the collected data, and by inputting the number of units for the term, the system will predict the amount of the tuition and fees.**

Keywords*— *Artificial Neural Network, Neural Network, tuition fee, tuition fee prediction, student, Computer Engineering, De La Salle University.

I. Introduction

Using the basic concepts of Machine Intelligence, we developed a system which predicts the tuition and fees of a university students assessment mainly based on the number of units the student is taking.
Curently, De La Salle University's My LaSalle (MLS) Portal has an online computing tool for tuition and fees. However, the system requires the student to pre-set initial information such as inputting all the subjects or courses taken. This can sometimes become too tedious listing down and trying to recall all the subjects the student enlisted for. Our system will only require the number of units taken by the student and will then predict the total cost for the tuiton and fees.
We will collect data from as much Enrollment Assessment Forms (EAFs) from different students as we can collect (including our own), taking note of the total number of units for the term, the total amount of the tuition and fees, and other pertinent values if needed. Then, we will load these data to our system as the Training Data. The EAFs that we will collect will only be from students taking up a bachelor's degree program in Computer Engineering. And, thus, our data and system application will only be limited to Computer Engineering students.

After programming the system, we will test it by giving a certain number of units and hopefully, the system can predict the cost of the tuition and fees with at least 80% accuracy.

II. Objectives

1) To create an Artificial Neural Network which will solve a problem either within the Philippines or specifically within the De La Salle University (DLSU) campus.
2) To collect a set of unique data (data from the EAFs) from BS-CPE students of DLSU.
3) To divide the data into Train, Validation and Test.
4) To produce a cost function plot.
5) To display the Mean-Squared Error (MSE) plots after testing.
6) To develop a system which has an accuracy of at least 80%.

III. Procedure

The sample data used for his project, as mentioned, were obtained from different Computer Engineering students. It was preferable to obtain different EAFs from different Computer Engineering years to provide better training data. Unfortunately,due to time constraints, the group was only able to acquire EAFs from immediate sources such as batchmates and upper batchmen. Since the EAFs included the tuition fee and the additional fees that make up the total amount, the group considered the value of the total amount instead of just the tuition fee. The data acquired was then transferred to individual Microsoft Excel files (One for the inputs and the other for the outputs) to be used as the data. The group decided to use the MATLAB desktop application and use the Neural Network Toolbox. The Neural Network Toolbox, specifically the Neural Fitting App, in MATLAB allowed the user create and train a neural network with ease. Since the group only managed to obtain 102 samples for data, the samples were divided to 60% - 20% - 20% for training, validation and testing respectively to avoid overfitting of data.

IV. Data and Results

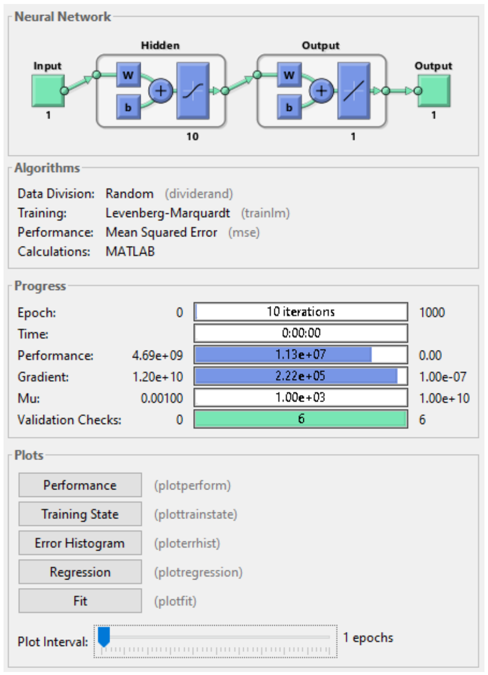
Note: A confusion matrix is not applicable for our system because it is a prediction system. Confusion matrices are usually used for classifications.

Generated Simple Script

```
% This script assumes these variables are
defined:
%
%   Inputs - input data.
%   Outputs - target data.

x = Inputs';
t = Outputs';

% Choose a Training Function
% For a list of all training functions
type: help nntrain
% 'trainlm' is usually fastest.
```



Neural Network

Algorithms

```
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MATLAB
```

Progress

```
Epoch:                0         10 iterations     1000
Time:                           0:00:00
Performance:          4.69e+09  1.13e+07          0.00
Gradient:             1.20e+10  2.22e+05          1.00e-07
Mu:                   0.00100   1.00e+03          1.00e+10
Validation Checks:    0         6                 6
```

Plots

```
Performance        (plotperform)
Training State     (plottrainstate)
Error Histogram    (ploterrhist)
Regression         (plotregression)
Fit                (plotfit)

Plot Interval:                     1 epochs
```

Fig. 1. Result of Training done using the Neural network tool

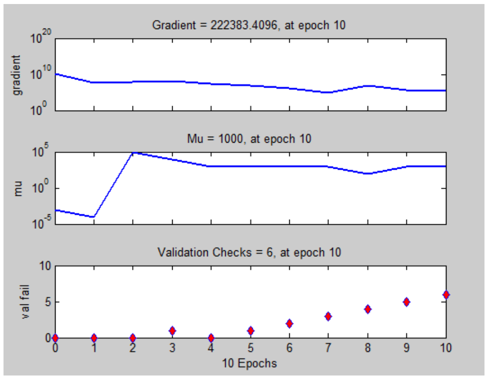


Fig. 2. Training State Plot

```
% 'trainbr' takes longer but may be better
for challenging problems.
% 'trainscg' uses less memory. NFTOOL
falls back to this in low memory
situations.
trainFcn = 'trainlm';  %
Levenberg-Marquardt

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training,
Validation, Testing
net.divideParam.trainRatio = 60/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 20/100;
```

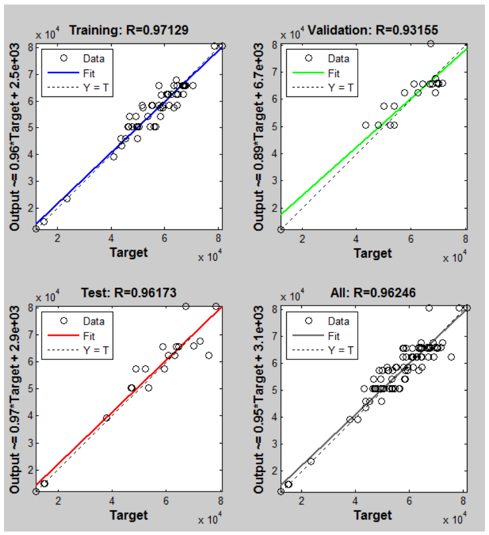


Fig. 3. Regression Plot

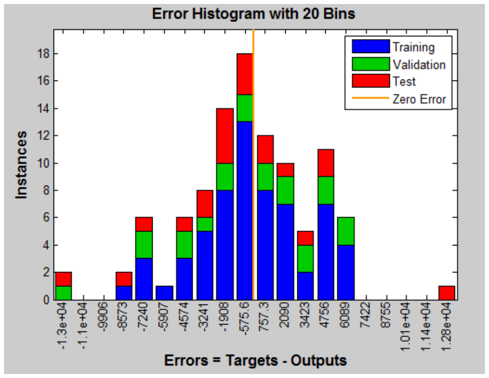


Fig. 4. Error Histogram

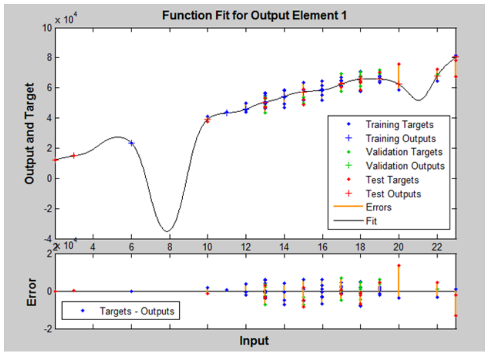


Fig. 5. Fit Plot

```
% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various
plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,x,t)
%figure, plotregression(t,y)
%figure, ploterrhist(e)
```

Generated Advanced Script

```
% This script assumes these variables are
defined:
%
%   Inputs - input data.
%   Outputs - target data.

x = Inputs';
t = Outputs';

% Choose a Training Function
% For a list of all training functions
type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better
for challenging problems.
% 'trainscg' uses less memory. NFTOOL
falls back to this in low memory
situations.
trainFcn = 'trainlm';  %
Levenberg-Marquardt

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output
Pre/Post-Processing Functions
% For a list of all processing functions
type: help nnprocess
net.input.processFcns =
{'removeconstantrows','mapminmax'};
net.output.processFcns =
{'removeconstantrows','mapminmax'};

% Setup Division of Data for Training,
Validation, Testing
% For a list of all data division
functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide
data randomly
net.divideMode = 'sample';  % Divide up
every sample
net.divideParam.trainRatio = 60/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 20/100;

% Choose a Performance Function
% For a list of all performance functions
type: help nnperformance
net.performFcn = 'mse';  % Mean squared
error

% Choose Plot Functions
% For a list of all plot functions type:
help nnplot
net.plotFcns = {'plotperform',
'plottrainstate', 'ploterrhist',
'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and
Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t  .* tr.valMask{1};
testTargets = t  .* tr.testMask{1};
trainPerformance = perform(net,trainTarget
s,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTarget
s,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various
plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,x,t)
%figure, plotregression(t,y)
%figure, ploterrhist(e)
```

```
% Deployment
% Change the (false) values to (true) to
enable the following code blocks.
if (false)
  % Generate MATLAB function for neural
network for application deployment
  % in MATLAB scripts or with MATLAB
Compiler and Builder tools, or simply
  % to examine the calculations your
trained neural network performs.
  genFunction(net,'myNeuralNetworkFunction');
  y = myNeuralNetworkFunction(x);
end
if (false)
  % Generate a matrix-only MATLAB function
for neural network code
  % generation with MATLAB Coder tools.
  genFunction(net,'myNeuralNetworkFunction'
,'MatrixOnly','yes');
  y = myNeuralNetworkFunction(x);
end
if (false)
  % Generate a Simulink diagram for
simulation or deployment with.
  % Simulink Coder tools.
  gensim(net);
end
```