



Ministry of Education and Science of the Russian Federation  
Peter the Great St. Petersburg State Polytechnic University  
Institute of Computer Science and Control Systems

**Control Systems and Technology Department**

## Final Project Report

Neural Network Approximation of Non-Linear Functions  
Course: Mathematical Modeling and Simulation

23 November, 2016

Student Group: 13541/8

\_\_\_\_\_ Christopher W. Blake

\_\_\_\_\_ Elizabeth Sekerina

Professor

\_\_\_\_\_ Rostov N.V

St. Petersburg  
2016

Contents

- Introduction ..... 3
- Background ..... 3
  - Neural Networks in Matlab..... 3
  - Hidden Layers ..... 3
  - Neurons Per Layer & Training Data ..... 4
  - Training Methods ..... 4
  - Non-Linear Functions..... 4
- Results..... 5
  - EQ 1:  $f(x) = x^2$  ..... 5
  - EQ 2:  $f(x) = \tan(x)$  ..... 6
  - EQ 3:  $f(x) = x^3 + 2 \cdot x^2 + \exp(x)$  ..... 7
- Conclusion..... 8
- References ..... 8
- Appendix 1 – Matlab Program Code ..... 9
  - Script: Program.m..... 9
  - Class: FunctionFitting.m..... 10

## Introduction

Topic 6 is about the approximation of a non-linear function using a neural network. The usage of neural networks for function approximation is well established, and it is also well known that a neural network, if properly designed, can estimate any function to any degree of needed accuracy.

The following functions will be modeled and approximated:

- $f(x) = x^2$
- $f(x) = \tan(x)$
- $f(x) = x^3 + 2x^2 + \exp(x)$

Within the Matlab toolkit, there exists many options for creating, training, testing, and utilizing a neural network. The previously mentioned functions are tested with two different neural network types, with one and two layers of hidden neurons, and with four different training methods. A comparison of these results is summarized, and the best configuration is recommended. The Matlab program code for this testing can be found in Appendix 1.

## Background

### Neural Networks in Matlab

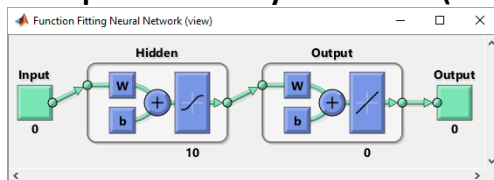
Two different built-in neural networks will be used to approximate the previously mentioned functions. Both network types are briefly described below.

- **feedforwardnet** – Feedforward neural network. This is the most common network type and is used as the basis for many other networks. The input values are connected to the first layer. There are one or more hidden layers, each of which is connected to each neuron in the neighboring layers. The final layer reduces to the desired number of outputs.
- **fitnet** – Function fitting neural network. This is a specialized version of a feedforward neural network.

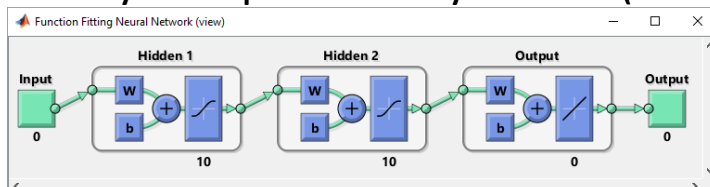
### Hidden Layers

Two layering schemes will be used. The two-layer network (one hidden layer) often referred to as a “Perceptron”, and the three-layer network (two hidden layers) often referred to as a “Multi-Layer Perceptron”. Images of both can be seen below. However, both networks will additionally be modified to work with more neurons per layer.

#### Perceptron: Two-Layer Network (one hidden layer)



#### Multi-Layer Perceptron: Three-Layer Network (two hidden layers)



## Neurons Per Layer & Training Data

The number of neurons per layer is very important for the function approximation by a neural network. Hence, the recommendation in Hagan et al. (1996), which helps to prevent overtraining of a network, will be used. The equation for this method is described by the below equation.  $n$  = number of inputs,  $M$  = number of hidden nodes (per layer).  $m$  = the number of training samples.

$$(n + 2)M + 1 < m$$

As such, two different number of neurons will be used.

1. The first number of neurons will be low: 3. Hence the training samples will be 11.
2. The second number of neurons method will simply be three times this minimum value. Hence 9 neurons and 29 training samples.

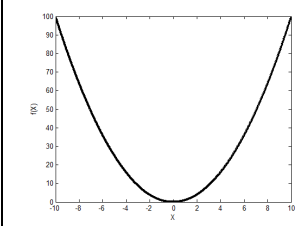
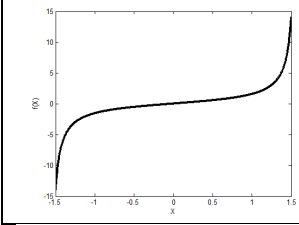
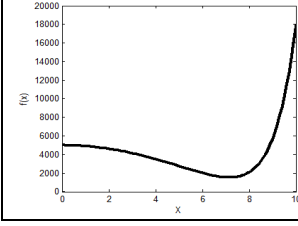
## Training Methods

There are 5 different training methods recommended function approximation section of the Matlab toolbox, all of which will be investigated. Below is a brief description of each.

1. **train** – Default normal method
2. **trainlm** – Levenberg-Marquardt backpropagation
3. **trainbr** – Bayesian regularization backpropagation
4. **trainscg** – Scaled conjugate gradient backpropagation
5. **trainrp** – Resilient backpropagation

## Non-Linear Functions

Three non-linear functions will be considered for this analysis. Each is shown below.

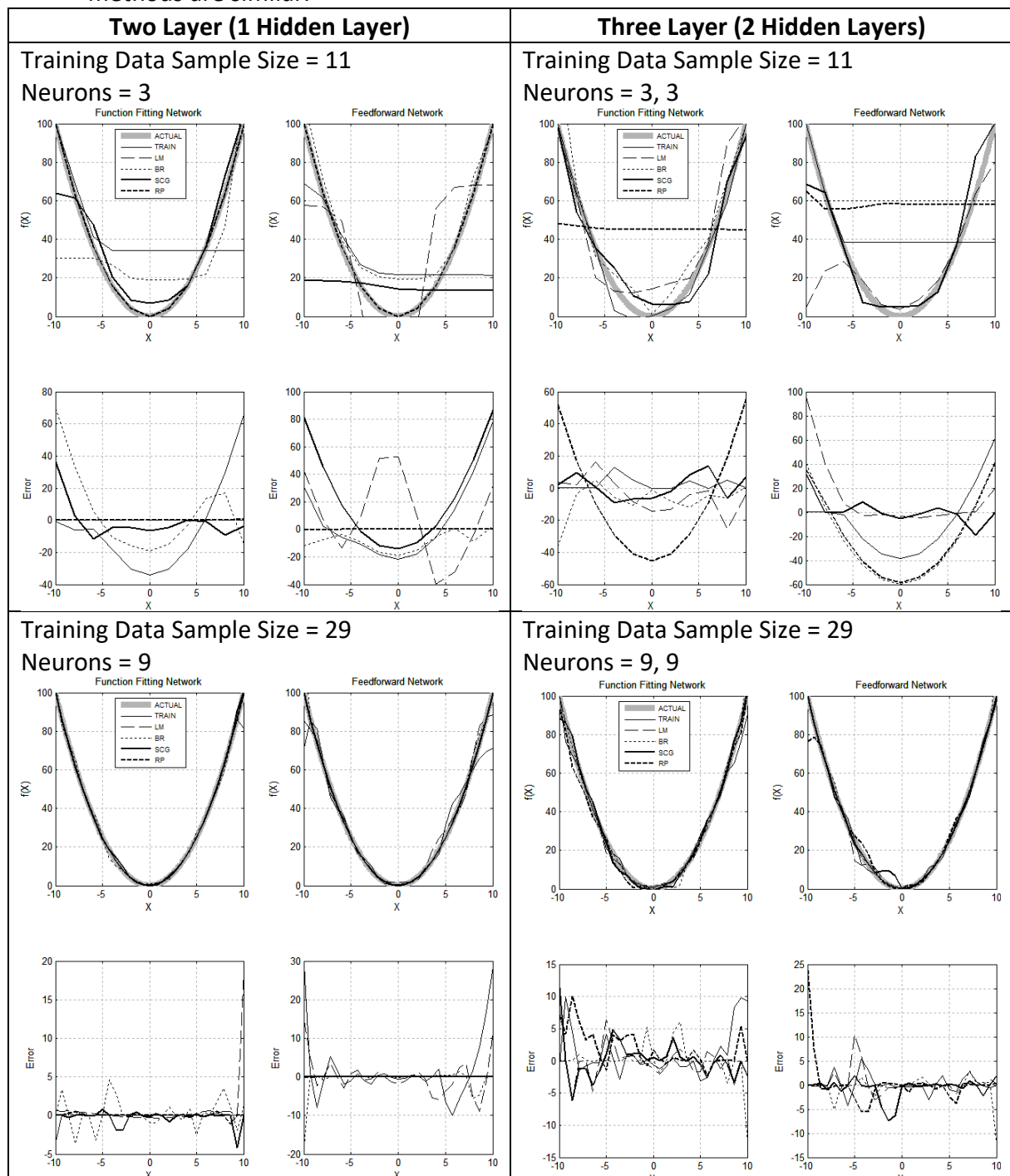
	<p><b>Equation:</b> <math>f(x) = x^2</math></p> <p><b>X Range:</b> -10 to 10</p>
	<p><b>Equation:</b> <math>f(x) = \tan(x)</math></p> <p><b>X Range:</b> -1.5 to 1.5</p>
	<p><b>Equation:</b> <math>f(x) = x^3 - 100*x^2 + \exp(x) + 5000</math></p> <p><b>X Range:</b> 0 to 10</p>

# Results

EQ 1:  $f(x) = x^2$

## Description

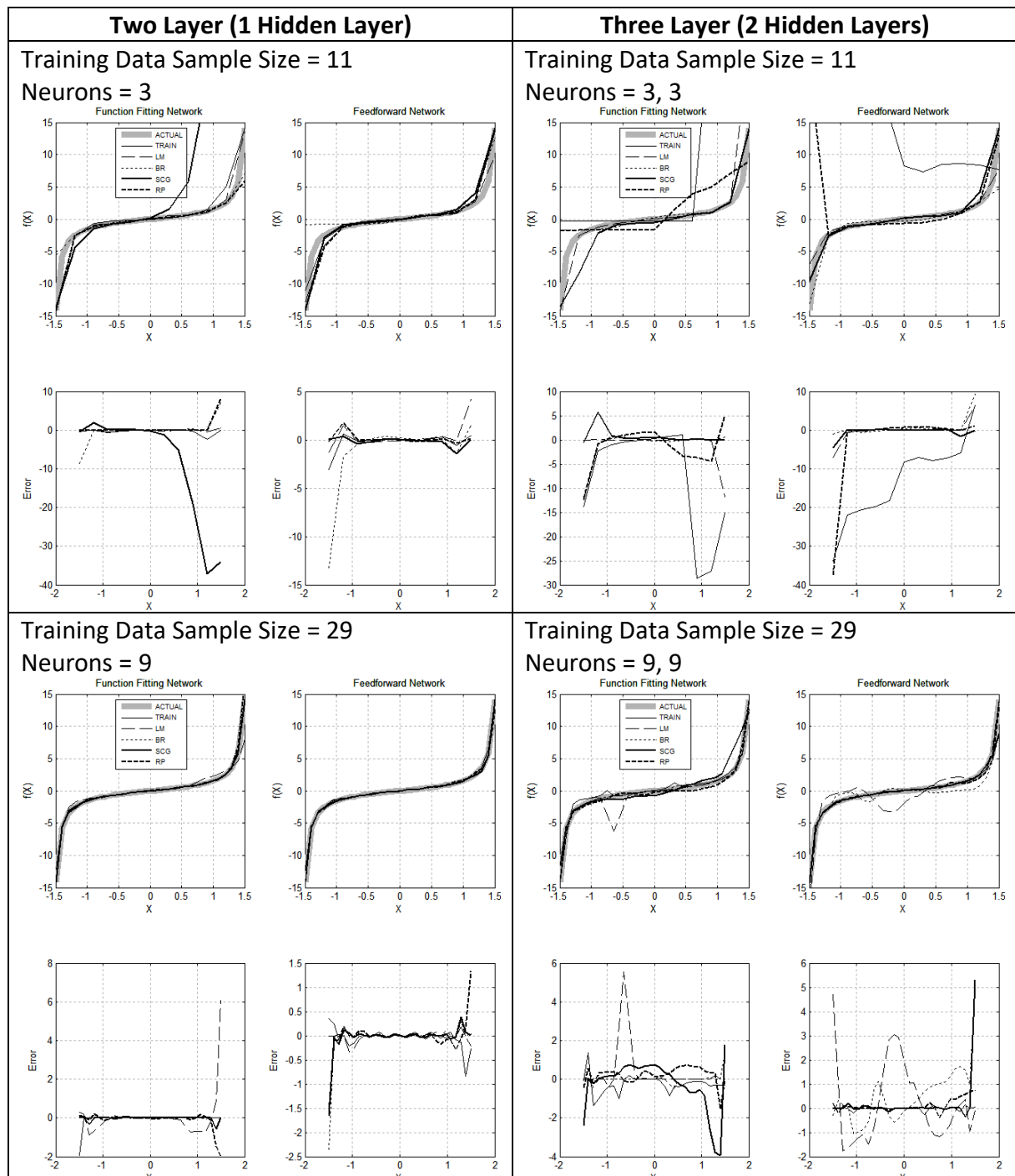
- Number of hidden layers** - The additional layer generally reduced error in the feedforward network, but increased error in the function fitting network.
- Number of neurons / sample data** – All cases are significantly more accurate.
- Network type** – The function fitting network in nearly all cases is more accurate.
- Training method** – If there is a low amount of sample data and a low number of neurons, the “RP” method is the most accurate. However, with more sample data, all methods are similar.



EQ 2:  $f(x) = \tan(x)$

**Description**

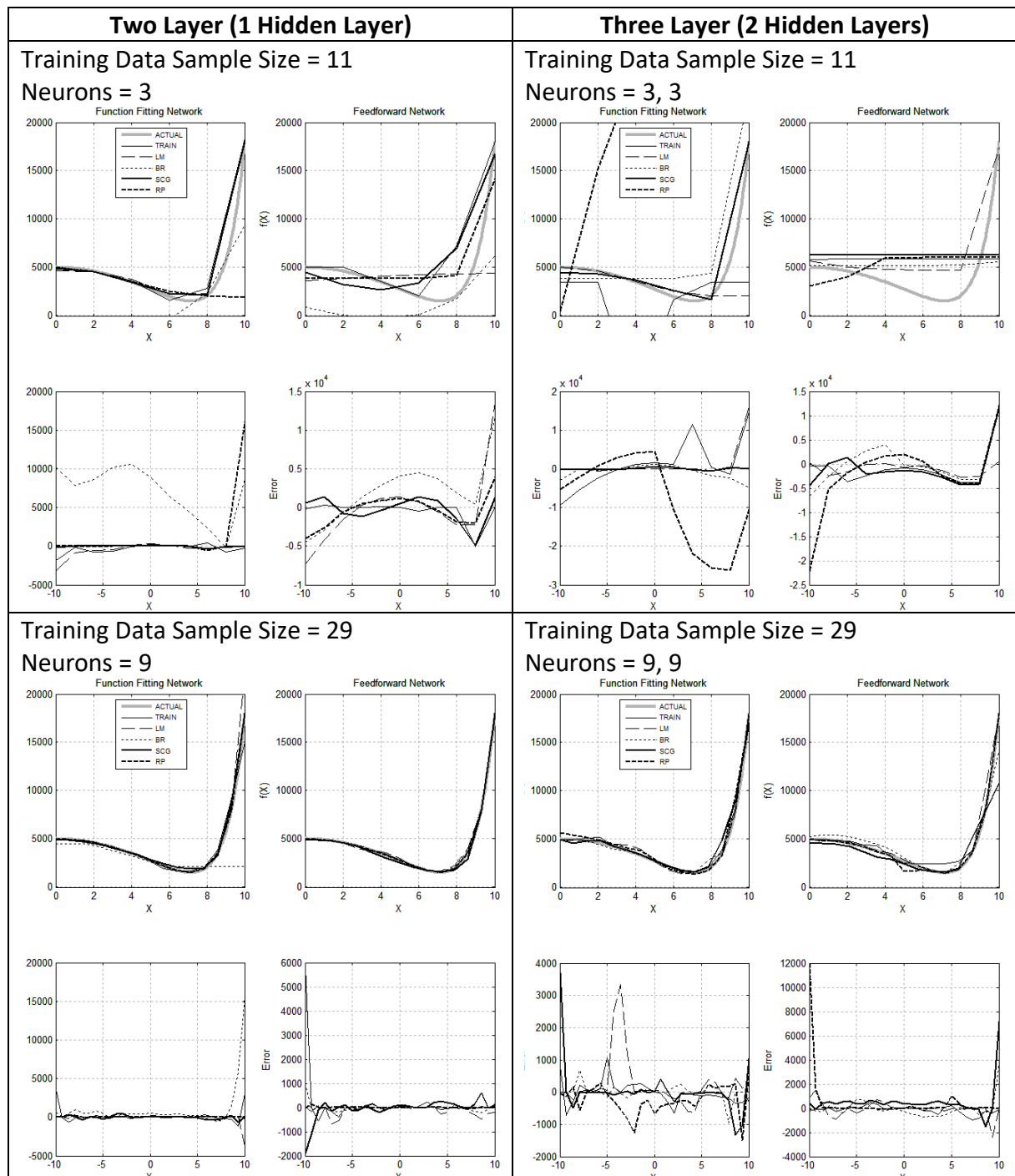
- Number of hidden layers** - The additional layer generally reduced error in the feedforward network, but increased error in the function fitting network.
- Number of neurons / sample data** – All cases are significantly more accurate.
- Network type** – The feedforward network in nearly all cases is more accurate.
- Training method** – With a single hidden layer, all training networks performed similarly. However, with two hidden layers, the “train” and “RP” methods had significant errors.



EQ 3:  $f(x) = x^3 + 2x^2 + \exp(x)$

**Description**

- Number of hidden layers** - The additional layer significantly increased the error.
- Number of neurons / sample data** – All cases are significantly more accurate.
- Network type** – With low sample data amounts, the function fitting network is more accurate. However, with more training data, the t feedforward is more accurate.
- Training method** – With low sample data amounts, all methods had significant error problems, especially in the later section ( $X > 8$ ). However, with more sample data, nearly all methods became significantly more accurate.



## Conclusion

Three different functions have been approximated using various different neural networks, configurations, and training methods. Below is a summary of the influence on each parameter when configuring a neural network.

1. **Number of hidden layers** – In two of the three functions, the additional layer generally reduced error in the feedforward network, but increased error in the function fitting network.
2. **Number of neurons / sample data** – All cases are significantly more accurate.
3. **Network type** – The function fitting network in nearly all cases is more accurate. However, with more training data the feedforward network may become more accurate.
4. **Training method** – With low sample data and number of neurons, all methods have error problems. However, the “RP” method is the most likely to remain accurate. With more sample data, all methods are similar.

The best network configuration in all cases is the Function Fitting Neural Network with a single hidden layer (9 neurons per layer). With this configuration, nearly all training methods perform accurately.

## References

1. Function Approximation and Nonlinear Regression (2016, November 24)  
<https://www.mathworks.com/help/nnet/function-approximation-and-nonlinear-regression.html>
2. Most, T.: Weimarer Optimierungs- und Stochastiktage 2.0 – 1./2. December 2005. Approximation of complex nonlinear functions by means of neural networks
3. Hagan, M. T. ; Demuth, H. B. ; Beale, M.: Neural Network Design. PWS Publishing Company, 1996



## Appendix 1 – Matlab Program Code

### Script: Program.m

```
% Final Project:
%   Neural Network Approximation of Non-Linear Functions
% Written By:
%   Christopher Blake
%   Elizabeth Sekerina
% Date:
%   24 November, 2016
clear; close all; clc;

%% General Configuration Properties
%Create same randomization seed point.
setdemorandstream(491218383)

%Change figure size
f1 = figure(1);
set(f1, 'Position', [50, 50, 750, 750]); % X from left, Y from bottom, Width, Height

%% Step 2: Create Neural Networks
% Function fitting neural network
    ShowInColum = 1;
    FunctionFitting.ShowNonLinearFunction(0.1, ShowInColum, 'Function Fitting Network'); %Show the
original non-linear function
    FunctionFitting.CreateAndShowNeuralNetwork(@fitnet, @train, 'k-', 1, ShowInColum); %solid
    FunctionFitting.CreateAndShowNeuralNetwork(@fitnet, @trainlm, 'k--', 1, ShowInColum); %dashed
    FunctionFitting.CreateAndShowNeuralNetwork(@fitnet, @trainbr, 'k:', 1, ShowInColum); %dotted
    FunctionFitting.CreateAndShowNeuralNetwork(@fitnet, @trainscg, 'k-', 2, ShowInColum); %thick
solid
    FunctionFitting.CreateAndShowNeuralNetwork(@fitnet, @trainrp, 'k--', 2, ShowInColum); %thick
dashed

% Feedforward neural network
    ShowInColum = 2;
    FunctionFitting.ShowNonLinearFunction(0.1, ShowInColum, 'Feedforward Network'); %Show the
original non-linear function
    FunctionFitting.CreateAndShowNeuralNetwork(@feedforwardnet, @train, 'k-', 1, ShowInColum);
    FunctionFitting.CreateAndShowNeuralNetwork(@feedforwardnet, @trainlm, 'k--', 1, ShowInColum);
    FunctionFitting.CreateAndShowNeuralNetwork(@feedforwardnet, @trainbr, 'k:', 1, ShowInColum);
    FunctionFitting.CreateAndShowNeuralNetwork(@feedforwardnet, @trainscg, 'k-', 2, ShowInColum);
    FunctionFitting.CreateAndShowNeuralNetwork(@feedforwardnet, @trainrp, 'k--', 2, ShowInColum);

%% Adjust format of plots
FunctionFitting.FormatSubplots({'ACTUAL', 'TRAIN', 'LM', 'BR', 'SCG', 'RP'}, 'north', 7);
```

## Class: FunctionFitting.m

```
classdef FunctionFitting
    %UNTITLED2 Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        %Subcharts
        fRows = 2;
        fCols = 2;

        %Display Range
        Xmin = -10;
        Xmax = 10;
        AxisLimits = [0,10, -1,20000];

        %Neural Networks
        NumberHiddenNodes = [3, 3];
        TrainingSetSize = 11;
        MaxEpochs = 500;
    end

    methods (Static)
        %This where the non-linear function is located.
        function Y = NonLinearFunction(X)
            %Y = X.^2;
            %Y = tan(X);
            Y = X.^3 - 100*X.^2 + exp(X) + 5000;
        end

        %DO NOT EDIT BELOW HERE
        function [X, Y] = GetTrainingData()
            this = FunctionFitting;

            % Determine step size
            step = (this.Xmax - this.Xmin)/(this.TrainingSetSize-1);

            % Generate training Datas
            X = this.Xmin:step:this.Xmax;
            Y = this.NonLinearFunction(X);
        end

        function ShowNonLinearFunction(step, position, ChartTitle)
            this = FunctionFitting;

            %Generate Data
            X = this.Xmin:step:this.Xmax;
            Y = this.NonLinearFunction(X);

            %Show plot
            subplot(this.fRows,this.fCols,position)
            plot(X,Y,'k-', 'LineWidth', 3, 'Color', [0.7, 0.7, 0.7]);
            title(ChartTitle);

            %Adjust axis
            axis(this.AxisLimits);

            %Allow others to draw on top
            grid on; hold on;
        end

        function CreateAndShowNeuralNetwork(NetworkType, TrainingMethod, LineStyle, LineWidth,
            Position)
            this = FunctionFitting;

            % Create the network
            net = NetworkType(this.NumberHiddenNodes);
            net.trainParam.epochs = this.MaxEpochs;

            %Train
            [X, Y] = this.GetTrainingData();
            [net, tr] = train(net,X,Y);

            % Get Results
            Y_net = net(X);

            %Display: Non-linear function approximation
            figure(1);
            subplot(this.fRows,this.fCols,Position)
            plot(X,Y_net,LineStyle,'LineWidth', LineWidth);
            xlabel('X');
            ylabel('f(X)');
            %legend('Y','Y_n');
```

```

%Display: MSE Error vs Epoch
% subplotPosition = ((2-1)*this.fCols) + Position; % Display in row 2
% subplot(this.fRows,this.fCols,subplotPosition);
% semilogy(tr.epoch, tr.tperf, LineStyle, 'LineWidth', LineWidth);
% xlabel('Epoch');
% ylabel('MSE');
% grid on; hold on;
%

%Display Y - Y_net
Y_actual = this.NonLinearFunction(X);
subplotPosition = ((2-1)*this.fCols) + Position; % Display in row 2
subplot(this.fRows,this.fCols,subplotPosition);
plot(X, Y_actual - Y_net, LineStyle, 'LineWidth', LineWidth); %Actual vs approximation
xlabel('X');
ylabel('Error');
grid on; hold on;

end
function FormatSubplots(LegendEntries, LegendLocation, FontSize)
this = FunctionFitting;

%Create Legend
subplot(this.fRows,this.fCols,1);
leg = legend(LegendEntries);
set(leg, 'FontSize', FontSize);
set(leg, 'Location', LegendLocation);
for r = 1:this.fRows
for c = 1:this.fCols
position = ((r-1)*this.fCols) + c ;
figure(1);
subplot(this.fRows,this.fCols,position);
set(gca, 'Xcolor', [0.7 0.7 0.7]);
set(gca, 'Ycolor', [0.7 0.7 0.7]);
Caxes = copyobj(gca,gcf);
set(Caxes, 'color', 'none', 'xcolor', 'k', 'xgrid', 'off', 'ycolor','k',
'ygrid','off');

%legend(LegendA,LegendB,LegendC);
%set(legend, 'location', 'northeast')

end
end
end
end
end
end
end
end

```