```
classdef neural_network

   properties
       weights={};
       biases={};
       features = [];
       target = [];
       learning_rate=0.01;
       epochs=100;
       hidden_units = 0;
       logistic = false;
       testing = false;
   end

   methods

   function self =
neural_network(input,hidden,features,target,learning_rate,epochs,logistic,testing

       self.weights = {rand(input,hidden),rand(hidden,1)};
       self.biases = {rand(1,hidden),rand(1,1)};
       self.features = features;
       self.target = target;
       self.learning_rate = learning_rate;
       self.epochs = epochs;
       self.hidden_units = hidden;
       self.logistic = logistic;
       self.testing = testing;

   end

   function [error,final_weights,final_activations,random_set] =
gradient_descent(obj)

       step = obj.epochs/10;
       error = zeros(obj.epochs,1);
       final_activations = zeros(obj.epochs,4);
       random_set = zeros(obj.epochs,3);
       final_weights = [];
```

Initialize weights and biases for testing purposes

```
       if obj.testing == true
           obj.features = [1,0];
           obj.target = 0;
           obj.weights = {[0,0.5;-1,1;0,1]',[0,1,-1]'};
           obj.biases = {[1;1;1]',1};
       end

       for epoch = 1:obj.epochs
```

```
        if epoch == 1
            counter = 0;
        end

            [nabla_w,nabla_b,err,activations,sample_set,counter]=
backprop(obj,counter);
            nabla_w = flip(nabla_w);
            nabla_b = flip(nabla_b);

            for element = 1:length(obj.weights)
                obj.weights{element} = obj.weights{element} -
(obj.learning_rate).*nabla_w{element}';
                obj.biases{element} = obj.biases{element} -
(obj.learning_rate).*nabla_b{element};
            end

        if (mod(epoch,step) == 0)
            fprintf('Epoch %2d: Error is: %2d \n',epoch,err);
            fprintf('Correct # of assignments this batch: %d /%d
\n',counter,step);

            if counter == step
                fprintf('Neural Network Training Complete! \n\n' );
                fprintf('Final Error Is: %e \n',
error(find(error,1,'last')));
                break
            end

            counter = 0;

        end

        if epoch == obj.epochs
            final_weights = obj.weights;
        end

        error(epoch) = err;
        final_activations(epoch,:) = cell2mat(activations);
        random_set(epoch,:) = sample_set;

        end

    end

    function
[nabla_w,nabla_b,err,final_activations,sample_set,counter] =
backprop(obj,counter)
        nabla_b = {};
        nabla_w = {};
        activation = [];
        activations = {};
        zs = {};
```

```
        if obj.testing == false
        %%Randomly sample feature set
            [feature,idx] = datasample(obj.features,1);
            target = obj.target(idx);
        else
            feature = obj.features;
            target = obj.target;
        end

        for layer = 1:length(obj.weights)

            if layer == 1
                z = (feature*obj.weights{layer})+obj.biases{layer};
                zs{end+1} = z;
                activation = neural_network.sigmoid(z);
                activations{end+1} = activation;

            else
                z
 =(activations{end}*obj.weights{layer})+obj.biases{layer};
                zs{end+1} = z;
                activation = neural_network.sigmoid(z);
                activations{end+1} = activation;

            end

        end

        for layer = length(obj.weights):-1:1

            if layer == length(obj.weights)

                if obj.logistic == true
                    delta =
 neural_network.log_partial_cost(activations{layer},target)*neural_network.sigmoid
                    [err] =
 neural_network.log_cost_function(activations{end},target);
                else
                    delta =
 neural_network.partial_cost(activations{layer},target)*neural_network.sigmoid_pri
                    [err] =
 neural_network.cost_function(activations{end},target);
                end
                    nabla_b{end+1} = delta;
                    nabla_w{end+1} = delta*activations{layer-1};
            else

                nabla_b{end+1} = delta*obj.weights{layer
+1}'.*neural_network.sigmoid_prime(activations{layer});
                nabla_w{end+1} = nabla_b{end}'*feature;
            end
        end

        [final_activations] = activations;
```

```
        [sample_set] = [feature, target];

        if (activations{end} > 0.95) && (target == 1) ||
 (activations{end} < 0.05) && (target == 0)
            counter = counter + 1;
        end

    end

    function plot_graph(obj,err)


 plot(1:1:find(err,1,'last'),err(1:find(err,1,'last')),'r'); hold on
            xlim([1,round(find(err,1,'last'))]);
            xlabel('Epoch'); ylabel('J_{\theta}'); legend('Cost
 Function','location','best');

    end

    end

    methods(Static)

        function hyp = sigmoid(z)
            hyp = 1.0./(1.0+exp(-z));
        end

        function delta = partial_cost(activations,y)
            delta = (activations-y);
        end

        function delta = log_partial_cost(a_x,target)
            delta = -(target/a_x)+((1-target)/(1-a_x));
        end

        function sig_prime = sigmoid_prime(activation)
            sig_prime = activation.*(1-activation);
        end

        function err = cost_function(activation,target)
            err = 0.5*(activation-target)^2;
        end
        function err = log_cost_function(activation,target)
            err = -target*log(activation)-(1-target)*log(1-
activation);
        end
    end

 end

Not enough input arguments.

Error in neural_network (line 19)
        self.weights = {rand(input,hidden),rand(hidden,1)};
```

Published with MATLAB® R2016a