# 重温手写数字分类-MATLAB面向对象实现

## MATLAB 面向对象简介

创建类的语法如下

```
1  classdef Point2D
2      properties
3          x
4          y
5      end
6      methods
7          function obj = Point2D(x0,y0)
8              obj.x = x0;
9              obj.y = y0;
10         end
11     end
12 end
```

这里，我们创建了二维点类，其属性值包含横纵坐标 x 和 y.

- 所有函数功能都放置在 methods ... end 之间，Line 7-10 给出的是类的构造函数，用以声明对象并初始化。
- 建立好类后，我们可在主程序中创建一个对象

  ```
  1  p = Point2D(0.5,0.5);
  ```

  输出 p 后可查看属性值

  ```
  1  p =
  2
  3    Point2D with properties:
  4
  5      x: 0.5000
  6      y: 0.5000
  ```

- 要特别说明的是：当添加其他方法 （即函数）时，我们常常会遇到修改属性值 x 和 y 的情况。例如添加方法

  ```
  1  function normalize(obj)
  2      r = sqrt(obj.x^2+obj.y^2);
  3      obj.x = obj.x/r;
  4      obj.y = obj.y/r;
  5  end
  ```

  系统会在 Line 5 显示提醒符号，对应的说明为：`Value class method that modifies the object must return the modified object`. 也就是说，默认情况下，我们建立的类是数值类，任何修改对象的操作都必须返回新的对象，即要把上面的方法修改为

```
function obj = normalize(obj)
    r = sqrt(obj.x^2+obj.y^2);
    obj.x = obj.x/r;
    obj.y = obj.y/r;
end
```

但此时查看对象

```
p = Point2D(0.5,0.5);
p.normalize();
disp(p)
```

可以发现，属性值仍未发生改变。这是因为，normalize 函数返回的对象其实是新的对象，我们必须重新创建一个对象，即改为

```
p = Point2D(0.5,0.5);
a = p.normalize();
disp(a)
```

- 如果希望在函数内修改类的属性值，而不必重新定义一个对象，那么可把类声明为 MATLAB 自带的句柄 ( handle) 类 （继承 handle 类）：

```
classdef Point2D < handle
```

此时不需要返回对象，任意修改操作都可改变属性值。

- 数值类与句柄类的区别在于内存的管理。上面的问题与 C 语言函数参数是值传递还是地址传递这一问题类似。值传递通常是在函数体的内部拷贝一个传入，函数体中的修改不会改变外部的实参。地址传递则不同，因为地址指向的内容是同一个东西。

# 建立类的框架

根据前面神经网络的讨论可知，SGD 训练的过程中涉及到两个重要步骤，即前向传播和后向传播。

- 前向传播是在给定网络参数的情况下，对输入进行预测。
- 后向传播是在给定小批量样本的输入和输出情况下，反向计算出所有层的神经元误差，进而得到梯度值，进行参数更新。

我们建立的类的框架如下

```
classdef Network < handle
    properties
        num_layers
        sizes
        biases
        weights
    end

    methods
        % constructor: initialization
        function obj = Network(sizes)
            obj.num_layers = length(sizes);
            obj.sizes = sizes;
```

```
            for s = 1:obj.num_layers-1 % 2,...,L
                obj.biases{s} = randn(sizes(s+1),1);
                obj.weights{s} = randn(sizes([s+1,s]));
            end
        end

        % feedforward
        [aL,a,z] = feedforward(obj,data_x);

        % backpropagation
        backprop(obj,mini_batch_y,a,z,eta);

        % train network by SGD
        SGD(obj, training_data_x, training_data_y, epochs, mini_batch_size,
eta, varargin);

        % evaluation of test_data
        [np,yp,y] = evaluate(obj,data_x,data_y);

    end  % end of methods

end
```

- 属性值的含义是显然的。我们直接给出了构造函数，其他方法只给出了函数声明。注意，在 MATLAB 中，若想类外实现方法，则必须建立 `@Network` 文件夹，并把类的定义文件和方法实现文件放入其中。
- 构造函数中我们对属性值进行了初始化，这里考虑的是一般情形，即隐藏层可以有多个层。
- feedforward 是前向传播方法，对输入 data_x 进行预测。注意，前向传播的参数由 obj.biases 和 obj.weights 获得。
- backprop 是反向传播方法，它接收小批量样本的输出、所有激活值和带权输入，然后反向计算神经元误差，计算梯度，并更新参数。

# 编写类的方法

## feedforward 方法

该函数比较简单，具体实现如下

```
function [aL,a,z] = feedforward(obj,data_x)
    a = cell(1,obj.num_layers);   % a1,...,aL
    z = cell(1,obj.num_layers-1); % z2,...,zL
    a{1} = data_x; aL = data_x;
    for s = 1:obj.num_layers-1
        w = obj.weights{s}; b = obj.biases{s};
        zs = w*aL+b; aL = sigmoid(zs);
        z{s} = zs; a{s+1} = aL;
    end
end
```

这里，aL 为最终的输出，a 和 z 则存储所有的激活值和带权输入。

后面我们不再逐一解释代码，熟悉面向过程的程序后，代码是自明的。

## backprop 方法

```
function backprop(obj,mini_batch_y,a,z,eta)
    % errors of neurons
    delta = cell(1,obj.num_layers-1);
    cost_a = cost_derivative(a{end}, mini_batch_y);
    delta{end} = cost_a.*sigmoid_prime(z{end});
    for i = 0:length(z)-2
        w3 = obj.weights{end-i};
        delta3 = delta{end-i};
        z2 = z{end-i-1};
        delta{end-i-1} = (w3'*delta3).*sigmoid_prime(z2);
    end
    % gradient descent: update weights and biases
    m = size(mini_batch_y,2);
    for level = 1:obj.num_layers-1 % 2,...,L
        delta2 = delta{level}; a1 = a{level};
        w = obj.weights{level};
        b = obj.biases{level};
        for i = 1:m   % loops of mini-batch data
            w = w - eta/m*delta2(:,i)*a1(:,i)';
            b = b - eta/m*delta2(:,i);
        end
        obj.weights{level} = w;
        obj.biases{level} = b;
    end

end % end of backprop

% derivative of cost function (w.r.t. a)
function cost_a = cost_derivative(aL, mini_batch_y)
    cost_a = aL - mini_batch_y;
end
```

## SGD 训练

```
function SGD(obj, training_data_x, training_data_y, epochs, mini_batch_size,
eta, varargin)
    n = size(training_data_x,2);
    batch_num = fix(n/mini_batch_size);    % number of mini-batches

    err = zeros(batch_num*epochs,1); st = 1;
    for ep = 1:epochs
        kk = randperm(n);  % for shuffling the training data
        for s = 1:batch_num
            % current mini-batch
            id = kk((s-1)*mini_batch_size+1 : s*mini_batch_size);
            mini_batch_x = training_data_x(:,id);
            mini_batch_y = training_data_y(:,id);

            % feedforward
            [aL,a,z] = obj.feedforward(mini_batch_x);

            % backpropagation
```

```
            obj.backprop(mini_batch_y,a,z,eta);

            % compute errors
            err(st) = 0.5*mean((aL(:)-mini_batch_y(:)).^2);
            st = st + 1;
        end

        % evaluation of test_data
        if ~isempty(varargin)
            test_data_x = varargin{1};
            test_data_y = varargin{2};
            ntest = size(test_data_x,2);
            np = obj.evaluate(test_data_x,test_data_y);
            fprintf('Epoch %2d :   %d / %d \n', ep, np, ntest);
        end
    end

    plot(err); ylim([0 0.5])
end % end of SGD
```

该函数中涉及到算法评估函数

```
function [np,yp,y] = evaluate(obj,data_x,data_y)
    test_data_yp = obj.feedforward(data_x);
    [~,yp] = max(test_data_yp,[],1);
    [~,y] = max(data_y,[],1);
    yp = yp'-1; y = y'-1;
    np = sum(yp==y); % number of correct predictions
end
```

其中，np 表示预测正确的数据个数，yp 是预测输出，y 是真实输出 （注意数字在 data_y 中是以 10 维的二进制向量表示的）。

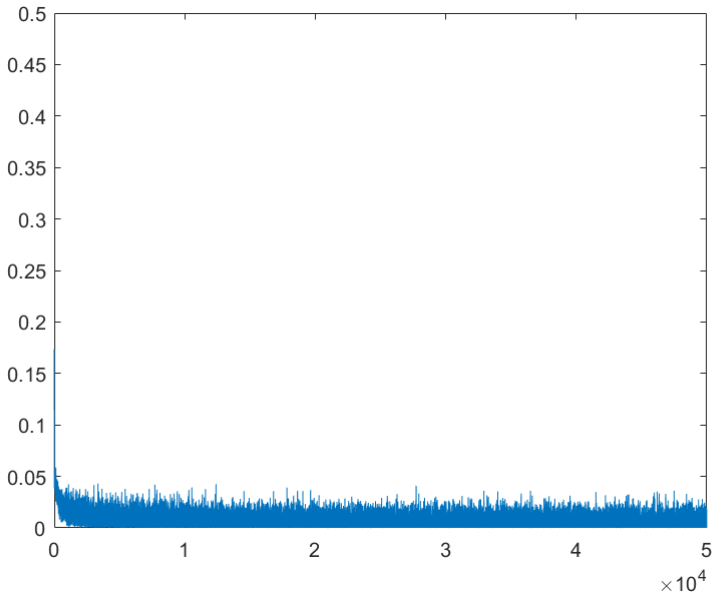
# 主程序

```
%% Load MNIST
% name_data_x, name_data_y, where name = training, validation, test
load mnistdata;

%% Parameters
sizes = [784 15 10];
epochs = 10;
mini_batch_size = 10;
eta = 3;

%% Create a Network object
net = Network(sizes);

%% Train network with SGD
net.SGD(training_data_x, training_data_y, epochs, mini_batch_size, eta, ...
        test_data_x, test_data_y);

%% Recognize handwritten digits
[np,y_p,y] = net.evaluate(validation_data_x,validation_data_y);
```

```
ratio = np/length(y);
fprintf('\n Recognize handwritten digits in validation_data \n');
fprintf(' Accuaracy = %.2f%% \n', ratio*100);
```

结果如下

```
Epoch  1 :   8882 / 10000
Epoch  2 :   9081 / 10000
Epoch  3 :   9170 / 10000
Epoch  4 :   9107 / 10000
Epoch  5 :   9220 / 10000
Epoch  6 :   9239 / 10000
Epoch  7 :   9189 / 10000
Epoch  8 :   9244 / 10000
Epoch  9 :   9264 / 10000
Epoch 10 :   9222 / 10000

 Recognize handwritten digits in validation_data
 Accuaracy = 92.07%
```




主程序中可设置三层以上的网络，如

```
sizes = [784 5 5 10];
```

结果为

```
Epoch  1 :   7497 / 10000
Epoch  2 :   8226 / 10000
Epoch  3 :   8242 / 10000
Epoch  4 :   8498 / 10000
Epoch  5 :   8493 / 10000
Epoch  6 :   8531 / 10000
Epoch  7 :   8606 / 10000
Epoch  8 :   8632 / 10000
Epoch  9 :   8608 / 10000
Epoch 10 :   8693 / 10000

 Recognize handwritten digits in validation_data
 Accuaracy = 85.82%
```

从精度和计算损耗方面来说，该案例三层网络的表现是最佳的。

---