Handwriting Number recognition neural network via Lisp *

刘光成
武汉大学
dormant@whu.edu.cn

Abstract

利用 Lisp 语言函数性编程的优点可以快捷的搭建 BP 神经网络。通过手写数字数据集的训练达到识别 28×28 像素图像中手写的数字的效果，当训练样本数量达到 500 以上时，正确率 >90%，下面按照构建 BP 神经网络的方法依次利用 Lisp 语言实现。

***K*eywords** Backpropagation · Lisp · Functional programming

1 Introduction

反向传播是一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法。该方法对网络中所有权重计算损失函数的梯度，这个梯度会回馈给最佳化方法，用来更新权值以最小化损失函数。BP 神经网络便是基于此方法来进行” 学习”，从而实现识别手写数字的功能。

2 Build a neuron

构建神经网络首先需要构建单独的神经元

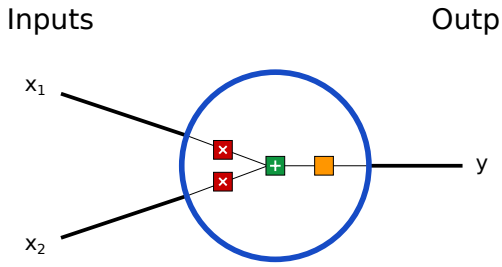

图 1: https://victorzhou.com/blog/intro-to-neural-networks/

* *author*: **刘光成. Handwriting Number recognition neural network via Lisp.**

在每个神经元中处理两个输入输出的过程可以分为两个部分

- 按照权重计算 x_1,x_2 并加上偏置输入得到 +
- 将上述过程计算得到的值经过**激活函数**得到输出

而常用的激活函数为 Sigmoid 函数 [以下简写为 $\sigma(x)$]

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{1}$$

其激活函数的目的是将所有的实数映射到 $(0,1)$ 之内，保证每一个神经元的输入输出均 $\in (0,1)$ 防止饱和。

```
(require math/array)
;Activation Function
(define (sigmoid x)
  (/ 1 (+ 1 (exp (- 0 x)))))
;A Neuron
(define (Neuron INPUT Weights bias)
  (let* ((s_l (array-map * Weights INPUT))
         (s (+ (array-all-sum s_l) bias)))
    (sigmoid s)))
```

这样便完成了单一神经元，理解了单一神经元的工作原理我们便可以将其链接起来组成神经网络。

3 Neuron Networks

当然神经网络可能有多层例如如下网络

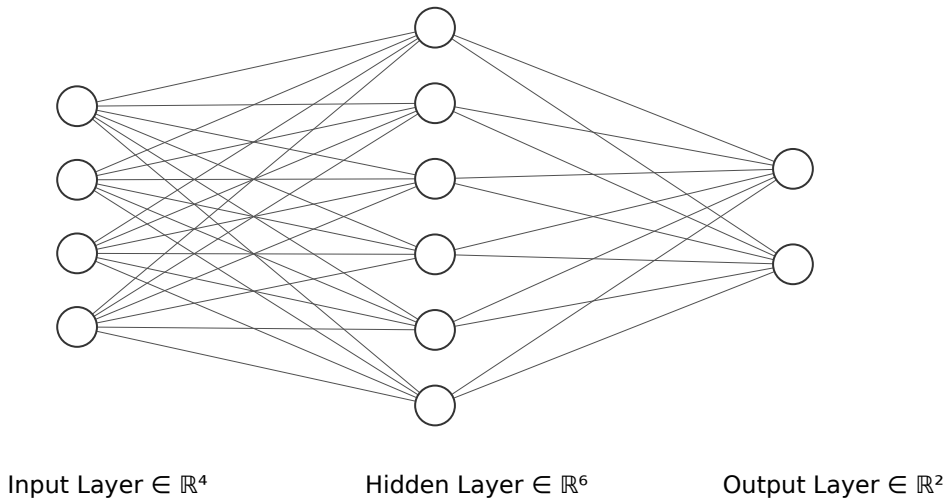

图 2: Simple Neuron Networks

这个网络有 4 个输入 (Input Layer)、一个包含 6 个神经元的隐藏层 (Hidden Layer)、包含 2 个神经元的输出层 (Output Layer)。

对于 Hidden Layer 的每个神经元会有 4 个上一层的神经元与其链接，其类比大脑中的神经元，链接强弱映射其边上的权重，便可以通过关联矩阵来描述输入输出的关系。

$$\underset{6\times1}{\mathrm{H}} = \underset{6\times4}{W_{ih}} \times \underset{4\times1}{\mathrm{I}} + \underset{6\times1}{\varepsilon_{ih}} \tag{2}$$

其中 ε 为偏置组成的列向量

即通过输入值即可计算出 Hidden Layer 的值为: $\underset{6\times1}{\mathrm{H}}$

同样的通过 Hidden Layer 也可以通过权重和偏置计算出 Output 的值

$$\underset{2\times1}{\mathrm{O}} = \underset{2\times6}{W_{ho}} \times \underset{6\times1}{\mathrm{H}} + \underset{2\times1}{\varepsilon_{ho}} \tag{3}$$

4 How Neuron Networks learn

要了解神经网络如何学习，需要明白一个问题-> 学习的最终效果是什么，对于识别手写数字，肯定是识别得到的数字与真实的数字越接近越好，故定义一个误差函数:

$$C(w,b) = \frac{1}{2n}\sum_{j=1}^{n}(y_{\mathrm{pred,j}} - y_{\mathrm{true,j}})^2 \tag{4}$$

最终的目标就是调整权重矩阵里面的 w 和偏置列向量里面的 b 使此函数最小这里采用最简单的梯度下降法

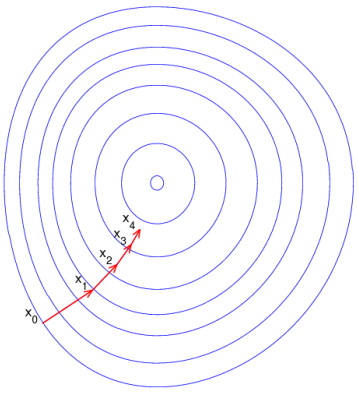

图 3: Gradient Descent

通过上图可以很容易的理解，在每一个点朝着该点梯度最大的方向移动一小段步长，然后一直移动即可找到局部的最优解

更新每一个 w, b 即可利用:

$$
\begin{aligned}
w_i &\to w_i - \eta \frac{\partial C}{\partial w_i} \\
b_i &\to b_i - \eta \frac{\partial C}{\partial b_i}
\end{aligned}
\tag{5}
$$

这里面的 η 为学习率 (learning rate)[也就是上面所说的每次移动的步长]

这里 $C(w, b)$ 对其中的 w_i 求偏导需要利用链式法则导出, 同样通过矩阵的形式将需要更新的关联矩阵表达出来:

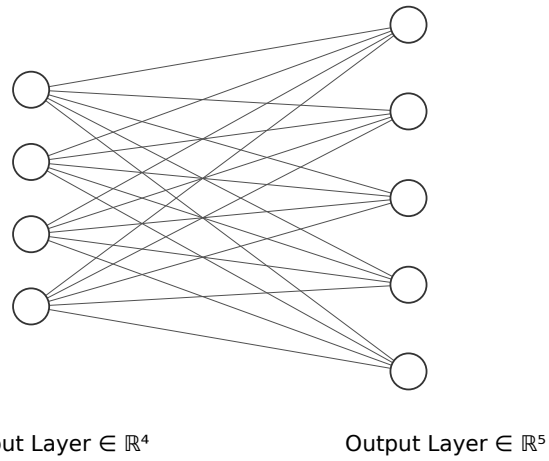

图 4: Simple Neuron Networks

仅写出邻近的两层之间的更新关系即可

利用链式法则将其写成三项:

$$
\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial y_{\text{pred},j}} \frac{\partial y_{\text{pred},j}}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{ij}} \tag{6}
$$

其中 $\frac{\partial C}{\partial y_{\text{pred},j}} = (y_{\text{pred,j}} - y_{\text{true,j}})$;$y_{\text{pred},j} = \sigma(z_j^L)$

故 $\frac{\partial y_{\text{pred},j}}{\partial z_j^L} = \sigma'(z_j^L) = \sigma(z_j^L)(1 - \sigma(z_j^L))$,$z_j^L = \sum_i w_{ji} a_i^{L-1}$

则可以得到整个权重矩阵的更新

$$
\underset{5\times4}{\frac{\partial C}{\partial \mathrm{W}}} = \underset{5\times1}{\frac{\partial C}{\partial y_{\text{pred}}}} \times \underset{5\times1}{\sigma'(z^L)} \times \underset{1\times4}{\text{transpose}(a^{L-1})} \tag{7}
$$

然后可得到更新后的权重矩阵:

$$
\underset{5\times4}{\mathrm{W}} = \underset{5\times4}{\mathrm{W}} - \eta \underset{5\times4}{\frac{\partial C}{\partial \mathrm{W}}} \tag{8}
$$

每两层之间都如此更新，如此” 学习” 即可最优化该权重矩阵。

5 Initialize Handwriting Number recognition neural network

构建的神经网络输入的节点应该有 784 个, 隐藏层设置为 200 个, 输出层节点设置为 10 个

[输入为 28px 乘以 28px 的灰度图，输出 0-9 十个数字]

```
;number of input, hidden and output layer nodes
(define input_nodes 784)
(define hidden_nodes 200)
(define output_nodes 10)
;learning rate is 0.1
(define lr 0.1)
```

值得注意的权重矩阵的初始化并不那么容易，Sigmoid(x) 限制了输出，导致在训练过程中权重矩阵很容易出现饱和的情况。Xavier 提出一种方法 [1] 来初始化，定义输入的神经元的数量为 n 则 W 的矩阵元从 0 为均值，标准差为 $\frac{1}{\sqrt{n}}$ 分布中随机取值:

其中高斯分布为:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{9}$$

奈何 racket 里面随机数只能生成整数无法从分布中取随机值，幸好 Xavier 给出了一个从均匀分布里面取值的方法:

对于 Sigmoid(x):

$$w \sim U([-\sqrt{\frac{96}{n_i + n_o}}, \sqrt{\frac{96}{n_i + n_o}}]) \tag{10}$$

w_i 从以上范围的均匀分布中取值便可得到初始化的 W 权重矩阵

首先通过'random' 函数生成一个可以生成两个范围内实数的程序;

```
;a:real with n decimal places
;b:real with n decimal places
;n:n decimal places
;suppose a<b
(define (random-real a b n)
  (let* ([low (exact-round (* (expt 10 n) a))]
         [high (exact-round(* (expt 10 n)  b))])
    (exact->inexact (/ (random low high) (expt 10 n)))))
```

则可初始化 wih,who:

```
;i_nodes:input_nodes
;o_nodes:output_nodes
(define (Gen_W_Matrix i_nodes o_nodes n)
  (let ([r (string->number (real->decimal-string (sqrt (/ 96 (+ i_nodes o_nodes))) n))])
  (build-matrix o_nodes i_nodes (lambda _ (random-real (- 0 r) r n)))))
```

其中real->decimal-string可以将实数转换成保留相应的小数的字符串的函数

至此已将权重矩阵初始化好，接下来就是构造我们的神经网络

6 Construct Handwriting Number recognition neural network

明确一下需求: 神经网络需要输入五个参数分别为 input nodes,hidden nodes,output nodes, 初始化后的 wih,who, 以及类中应该包含两个函数`train()`和`query()`其中`train()`包含两个参数: 训练集与其对应的准确值，通过梯度下降算法更新权重矩阵，`query()`需要通过输入测试值输出神经网络的预测值。

神经网络类的实现:

```
(define neuralNetwork%
    (class object%
      ;initialization argument
      (init-field inputnodes hiddennodes outputnodes learning_rate wih who)
      ;superclass initialization
      (super-new)
      ;train function
      (define/public (train input_list target_list)
        (let* (
            ;calculate inputs into hidden layer
            [hidden_inputs (matrix* wih input_list)]
            ;calculate outputs emerging from hidden layer
            [hidden_outputs (matrix-map (lambda (x) (sigmoid x)) hidden_inputs)]
            ;calculate inputs into output layer
            [final_inputs (matrix* who hidden_outputs)]
            ;calculate outputs emerging from output layer
            [final_outputs (matrix-map (lambda (x) (sigmoid x)) final_inputs)]
            ;output layer error is the (target - actual) (matrix sub)
            [output_errors (matrix- target_list final_outputs)]
            ;hidden layer error
            [hidden_errors (matrix* (matrix-transpose who) output_errors)]
            ;calculate first term of the dC/dW like (7)
            [_who (matrix-map (lambda (x y z) (* x y z)) output_errors final_outputs (matrix-map (
    lambda (x) (- 1 x)) final_outputs))]
            [_wih (matrix-map (lambda (x y z) (* x y z)) hidden_errors hidden_outputs (matrix-map (
    lambda (x) (- 1 x)) hidden_outputs))]
            [Dwih (matrix* _wih (matrix-transpose input_list))]
            [Dwho (matrix* _who (matrix-transpose hidden_outputs))])
            ;calculate second term of the dC/dW like (7)
            (begin
                (set! wih (matrix+ wih (matrix-scale Dwih learning_rate)))
                (set! who (matrix+ who (matrix-scale Dwho learning_rate))))))
      ;query function()
      (define/public (query input_list)
        (let* ([hidden_inputs (matrix* wih input_list)]
               [hidden_outputs (matrix-map (lambda (x) (sigmoid x)) hidden_inputs)]
               [final_inputs (matrix* who hidden_outputs)]
               [final_outputs (matrix-map (lambda (x) (sigmoid x)) final_inputs)])
           final_outputs))))
```

并实例化一个神经网络:

```
;number of input, hidden and output nodes
(define input_nodes 784)
(define hidden_nodes 200)
(define output_nodes 10)
;learning rate is 0.1
```

```
(define lr 0.1)
(define n (new neuralNetwork%
               [inputnodes input_nodes]
               [hiddennodes hidden_nodes]
               [outputnodes output_nodes]
               [learning_rate lr]
               ;4:keep to 4 decimal places
               [wih (Gen_W_Matrix input_nodes hidden_nodes 4)]
               [who (Gen_W_Matrix hidden_nodes output_nodes 4)]))
```

7 Train Handwriting Number recognition neural network

训练神经网络类

利用训练的数据集来自http://yann.lecun.com/exdb/mnist/, 其中采用的手写照片为 28px × 28px, 并且每一个像素点对应 0-255 的灰度颜色

如图:

图 5: 3

图 6: 4

图 7: 5

图 8: Handwritting Number

训练数据集每一行的第一列提供该数字的精确值, 剩下的 784(=28×28) 提供每个像素的灰度值

True Number	(1,1)	···	(28,28)
5	0	···	0
0	0	···	0
4	0	···	0

表 1: datasets

采用csv-reading库来读取数据:

```
(define train_datasets
  (make-csv-reader
   (open-input-file "./mnist_train_100.csv")
   ;delete the whitespace
   '((strip-leading-whitespace?  . #t)
     (strip-trailing-whitespace? . #t))))
;transform csv->list
(define t_d (csv->list train_datasets))
```

由于转换为列表后每一行的数值是采用的字符串的方式存储的，而之后需要通过行数来访问每一行的数据并将其转换为数字:

```
;Access data via index
(define (record training_data_list ref)
  (map (lambda (x) (string->number x)) (list-ref training_data_list ref)))
```

注意到每一行的第一个数字为训练集的真实值，剩下的为图片的 784 个像素中的灰度值故通过两个程序将每一行其中的数值转换为神经网络可以识别的输入输出数据，并且对于灰度值我们通过/255 * 0.99 + 0.01 的方式将其归一化 [不能让输入值中出现饱和]，并将输入和真实值转化为列向量:

```
;t_d_everyline->input_datasets
(define (data_processing->input t_d_line)
  (let* (;Get all gray value of the picture
         [true_data_list (rest t_d_line)]
         ;Normalized
         [inputs_list (map (lambda (x) (+ (* (/ x 255.0) 0.99) 0.01)) true_data_list)]
         ;List->col-matrix
         [inputs (->col-matrix (list->array inputs_list))])
  inputs))
;t_d_everyline->output_true
(define (data_processing->true t_d_line)
  (let* (;Get true value of the number
         [true_number (first t_d_line)]
         ;Generate output_list
         [true_list (list-set zeros true_number 0.99)]
         ;List->col-matrix
         [true (->col-matrix (list->array true_list))])
    true))
;Define zeros_list
(define zeros (build-list output_nodes (lambda _ 0.01)))
```

至此数据的预处理就全部完成了，将每一行的数据扔到神经网络类里面进行训练即可:

```
;Define the length of the data_sets
(define t_d_l (length t_d))
;Define how many times to train this training sets
(define epochs 5)
;Train
(for ([i (build-list epochs values)])
    (for ([j (build-list t_d_l values)])
    (begin
        (let* ([every_line (record t_d j)])
        [send n train
            (data_processing->input every_line)
            (data_processing->true every_line)]))))
```

8 Test Handwriting Number recognition neural network

通过test_datasets来测试我们的神经网络:

在此之前先通过一个程序将每一列的 784 像素的灰度值转换为可视化图片;

首先将每一列行的数据恢复为 28×28 的矩阵

```
;List->Matrix
(define (List->Image_Matrix t_d_line)
```

```
  (let* (;Get all gray value of the picture
         [true_data_array (rest t_d_line)]
         [true_data_matrix (list->matrix 28 28 true_data_array)])
    (matrix-transpose true_data_matrix)))
```

由于是灰度图故可以将每一数据点对应到一个灰度颜色上:

```
;map a value to a color
(define (cmap v)
  (color (floor (- 255 v))
         (floor (- 255 v))
         (floor (- 255 v))))
```

再将矩阵中的值转换为图片即可:

```
;Matrix->Image
(require 2htdp/image)
(define (M->Image M)
    (apply
     above
     (for/list ([y (in-range 28)])
      (apply
       beside
      (for/list ([x (in-range 28)])
        (rectangle 10 10 'solid (cmap (array-ref M (vector x y)))))))))
```

并且对于输出，由于神经网络输出为一列向量，而其预测的数字为其中最大值的 index:

```
;Find the index via value
(define (index-of lst ele)
  (let loop ((lst lst)
             (idx 0))
    (cond ((empty? lst) #f)
          ((equal? (first lst) ele) idx)
          (else (loop (rest lst) (add1 idx))))))
;col-Matrix->Index of max value
(define (output_list->number o_list)
  (let* (;col-matrix->list
         ;list->array
         [o_l (matrix->list o_list)]
         [o_array (list->array o_l)]
         ;Find max value of the o_list
         [Max_V (array-all-max o_array)]
         ;Find the index via value
         [index (index-of o_l Max_V)])
    index))
```

定义测试程序:

```
(define (test test_datasets ref)
  (let ([pred_number (output_list->number (send n query (data_processing->input (record test_datasets
    ref))))])
    (place-image (text (number->string pred_number) 32 "black") 20 20 (M->Image (List->Image_Matrix (
    record test_datasets ref))))))
```

将图片与预测值对比发现训练 100 个数据集训练 5 次，正确率高达 90%

展示神经网络预测结果和真实图片对比 [左上角为神经网络预测结果]:

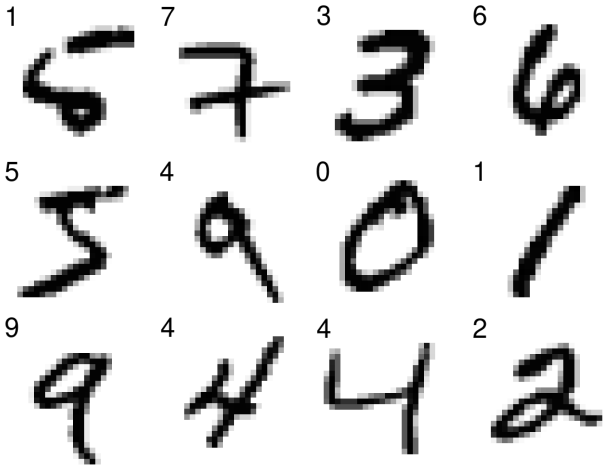

图 9: Comparison between predicted value and actual picture

9 Conclusion

绘制出训练次数与预测误差曲线

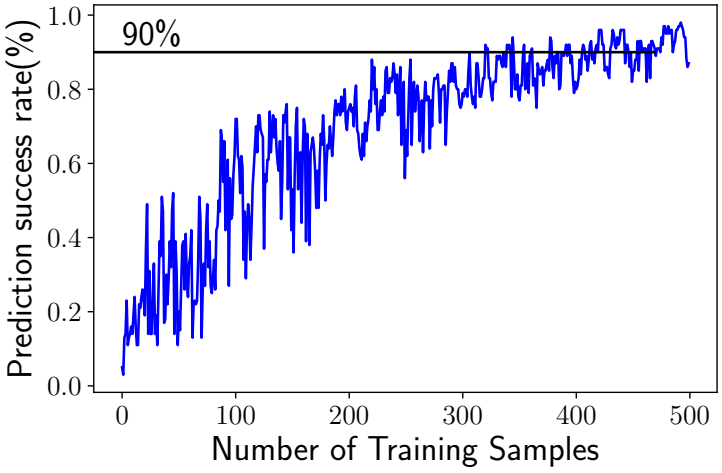

图 10: Comparison between predicted value and actual picture

会发现当样本数量越多神经网络预测的结果越好，当训练的样本数量 >500 次时效果基本满足要求，识别正确率 >90%

Acknowledgments

Datasets are supported in `http://yann.lecun.com/exdb/mnist/`.

The basic knowledge about neural networks comes from a book `https://www.amazon.com/Make-Your-Own-Neural-Network/dp/1530826608/r`.

The required racket package:

```
(require csv-reading)
(require racket/base)
(require racket/class)
(require math/array)
(require math/matrix)
(require math/distributions)
(require 2htdp/image)
```

参考文献

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

10 Appendix

```
(require csv-reading)
(require racket/base)
(require racket/class)
(require math/array)
(require math/matrix)
(require math/distributions)
(require 2htdp/image)

;Activation Function
(define (sigmoid x)
  (/ 1 (+ 1 (exp (- 0 x)))))

;a:real with n decimal places
;b:real with n decimal places
;n:n decimal places
;suppose a<b
(define (random-real a b n)
  (let* ([low (exact-round (* (expt 10 n) a))]
         [high (exact-round(* (expt 10 n)  b))])
    (exact->inexact (/ (random low high) (expt 10 n)))))

;i_nodes:input_nodes
;o_nodes:output_nodes
(define (Gen_W_Matrix i_nodes o_nodes n)
  (let ([r (string->number (real->decimal-string (sqrt (/ 96 (+ i_nodes o_nodes))) n))])
  (build-matrix o_nodes i_nodes (lambda _ (random-real (- 0 r) r n)))))

```

```
(define neuralNetwork%
    (class object%
      ;initialization argument
      (init-field inputnodes hiddennodes outputnodes learning_rate wih who)
      ;superclass initialization
      (super-new)
      ;train function
      (define/public (train input_list target_list)
        (let* (
            ;calculate inputs into hidden layer
            [hidden_inputs (matrix* wih input_list)]
            ;calculate outputs emerging from hidden layer
            [hidden_outputs (matrix-map (lambda (x) (sigmoid x)) hidden_inputs)]
            ;calculate inputs into output layer
            [final_inputs (matrix* who hidden_outputs)]
            ;calculate outputs emerging from output layer
            [final_outputs (matrix-map (lambda (x) (sigmoid x)) final_inputs)]
            ;output layer error is the (target - actual) (matrix sub)
            [output_errors (matrix- target_list final_outputs)]
            ;hidden layer error
            [hidden_errors (matrix* (matrix-transpose who) output_errors)]
            ;calculate first term of the dC/dW like (7)
            [_who (matrix-map (lambda (x y z) (* x y z)) output_errors final_outputs (matrix-map (
    lambda (x) (- 1 x)) final_outputs))]
            [_wih (matrix-map (lambda (x y z) (* x y z)) hidden_errors hidden_outputs (matrix-map (
    lambda (x) (- 1 x)) hidden_outputs))]
            [Dwih (matrix* _wih (matrix-transpose input_list))]
            [Dwho (matrix* _who (matrix-transpose hidden_outputs))])
            ;calculate second term of the dC/dW like (7)
            (begin
                (set! wih (matrix+ wih (matrix-scale Dwih learning_rate)))
                (set! who (matrix+ who (matrix-scale Dwho learning_rate))))))
      ;query function()
      (define/public (query input_list)
        (let* ([hidden_inputs (matrix* wih input_list)]
               [hidden_outputs (matrix-map (lambda (x) (sigmoid x)) hidden_inputs)]
               [final_inputs (matrix* who hidden_outputs)]
               [final_outputs (matrix-map (lambda (x) (sigmoid x)) final_inputs)])
           final_outputs))))

;number of input, hidden and output nodes
(define input_nodes 784)
(define hidden_nodes 200)
(define output_nodes 10)
;learning rate is 0.1
(define lr 0.1)
(define n (new neuralNetwork%
               [inputnodes input_nodes]
               [hiddennodes hidden_nodes]
               [outputnodes output_nodes]
               [learning_rate lr]
               ;4:keep to 4 decimal places
               [wih (Gen_W_Matrix input_nodes hidden_nodes 4)]
               [who (Gen_W_Matrix hidden_nodes output_nodes 4)]))

```

```
(define train_datasets
  (make-csv-reader
   (open-input-file "./mnist_train_100.csv")
   ;delete the whitespace
   '((strip-leading-whitespace?  . #t)
     (strip-trailing-whitespace? . #t))))
;transform csv->list
(define t_d (csv->list train_datasets))

;Access data via index
(define (record training_data_list ref)
  (map (lambda (x) (string->number x)) (list-ref training_data_list ref)))

;t_d_everyline->input_datasets
(define (data_processing->input t_d_line)
  (let* (;Get all gray value of the picture
         [true_data_list (rest t_d_line)]
         ;Normalized
         [inputs_list (map (lambda (x) (+ (* (/ x 255.0) 0.99) 0.01)) true_data_list)]
         ;List->col-matrix
         [inputs (->col-matrix (list->array inputs_list))])
  inputs))
;t_d_everyline->output_true
(define (data_processing->true t_d_line)
  (let* (;Get true value of the number
         [true_number (first t_d_line)]
         ;Generate output_list
         [true_list (list-set zeros true_number 0.99)]
         ;List->col-matrix
         [true (->col-matrix (list->array true_list))])
    true))
;Define zeros_list
(define zeros (build-list output_nodes (lambda _ 0.01)))

;Define the length of the data_sets
(define t_d_l (length t_d))
;Define how many times to train this training sets
(define epochs 5)
;Train
(for ([i (build-list epochs values)])
    (for ([j (build-list t_d_l values)])
    (begin
        (let* ([every_line (record t_d j)])
        [send n train
            (data_processing->input every_line)
            (data_processing->true every_line)])))))

;List->Matrix
(define (List->Image_Matrix t_d_line)
  (let* (;Get all gray value of the picture
         [true_data_array (rest t_d_line)]
         [true_data_matrix (list->matrix 28 28 true_data_array)])
    (matrix-transpose true_data_matrix)))

;map a value to a color
```

```
(define (cmap v)
  (color (floor (- 255 v))
         (floor (- 255 v))
         (floor (- 255 v))))

;Matrix->Image
(define (M->Image M)
    (apply
     above
     (for/list ([y (in-range 28)])
      (apply
       beside
      (for/list ([x (in-range 28)])
        (rectangle 10 10 'solid (cmap (array-ref M (vector x y)))))))))

;Find the index via value
(define (index-of lst ele)
  (let loop ((lst lst)
             (idx 0))
    (cond ((empty? lst) #f)
          ((equal? (first lst) ele) idx)
          (else (loop (rest lst) (add1 idx))))))
;col-Matrix->Index of max value
(define (output_list->number o_list)
  (let* (;col-matrix->list
         ;list->array
         [o_l (matrix->list o_list)]
         [o_array (list->array o_l)]
         ;Find max value of the o_list
         [Max_V (array-all-max o_array)]
         ;Find the index via value
         [index (index-of o_l Max_V)])
    index))

(define (test test_datasets ref)
  (let ([pred_number (output_list->number (send n query (data_processing->input (record test_datasets
    ref))))])
    (place-image (text (number->string pred_number) 32 "black") 20 20 (M->Image (List->Image_Matrix (
    record test_datasets ref))))))
```