Homework 4 for Pattern Recognition

Fan JIN (2015011506)

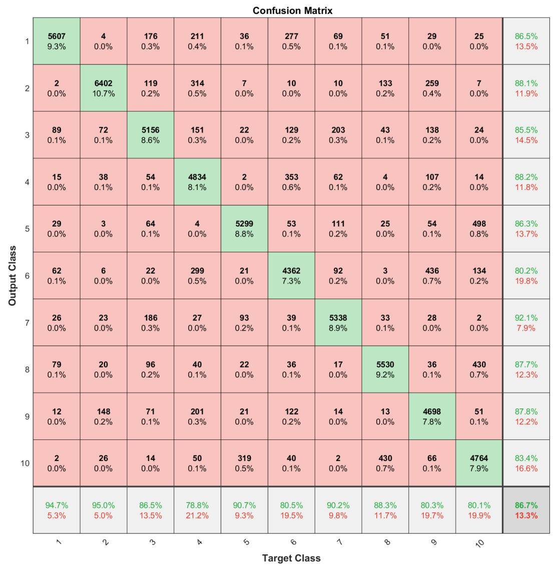
April 22, 2018

Question 1

Visualization

Figure 1: A glance at all the 10 categories

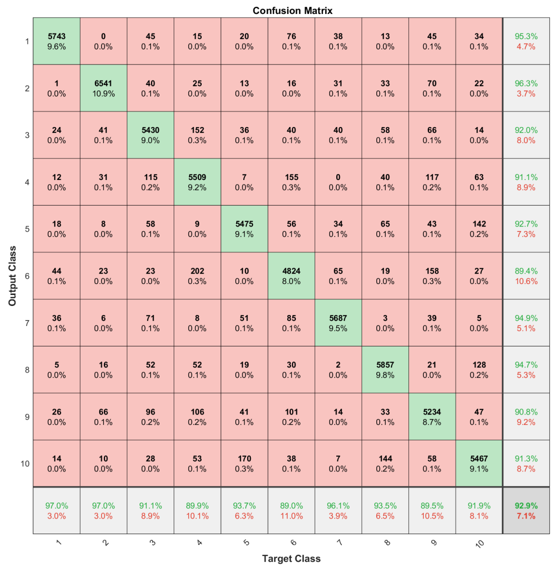
Confusion Matrices



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	5607 9.3%	4 0.0%	176 0.3%	211 0.4%	36 0.1%	277 0.5%	69 0.1%	51 0.1%	29 0.0%	25 0.0%	86.5% 13.5%
2	2 0.0%	6402 10.7%	119 0.2%	314 0.5%	7 0.0%	10 0.0%	10 0.0%	133 0.2%	259 0.4%	7 0.0%	88.1% 11.9%
3	89 0.1%	72 0.1%	5156 8.6%	151 0.3%	22 0.0%	129 0.2%	203 0.3%	43 0.1%	138 0.2%	24 0.0%	85.5% 14.5%
4	15 0.0%	38 0.1%	54 0.1%	4834 8.1%	2 0.0%	353 0.6%	62 0.1%	4 0.0%	107 0.2%	14 0.0%	88.2% 11.8%
5	29 0.0%	3 0.0%	64 0.1%	4 0.0%	5299 8.8%	53 0.1%	111 0.2%	25 0.0%	54 0.1%	498 0.8%	86.3% 13.7%
6	62 0.1%	6 0.0%	22 0.0%	299 0.5%	21 0.0%	4362 7.3%	92 0.2%	3 0.0%	436 0.7%	134 0.2%	80.2% 19.8%
7	26 0.0%	23 0.0%	186 0.3%	27 0.0%	93 0.2%	39 0.1%	5338 8.9%	33 0.1%	28 0.0%	2 0.0%	92.1% 7.9%
8	79 0.1%	20 0.0%	96 0.2%	40 0.1%	22 0.0%	36 0.1%	17 0.0%	5530 9.2%	36 0.1%	430 0.7%	87.7% 12.3%
9	12 0.0%	148 0.2%	71 0.1%	201 0.3%	21 0.0%	122 0.2%	14 0.0%	13 0.0%	4698 7.8%	51 0.1%	87.8% 12.2%
10	2 0.0%	26 0.0%	14 0.0%	50 0.1%	319 0.5%	40 0.1%	2 0.0%	430 0.7%	66 0.1%	4764 7.9%	83.4% 16.6%
	94.7% 5.3%	95.0% 5.0%	86.5% 13.5%	78.8% 21.2%	90.7% 9.3%	80.5% 19.5%	90.2% 9.8%	88.3% 11.7%	80.3% 19.7%	80.1% 19.9%	86.7% 13.3%

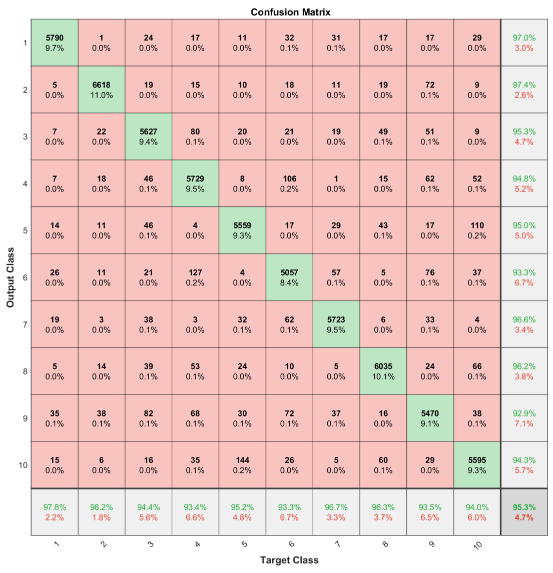
Figure 2: Confusion matrix with 5 hidden nodes



Confusion Matrix

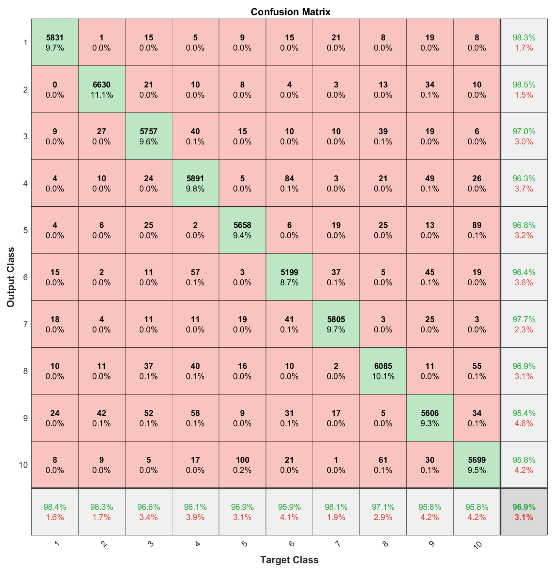
Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	5743 9.6%	0 0.0%	45 0.1%	15 0.0%	20 0.0%	76 0.1%	38 0.1%	13 0.0%	45 0.1%	34 0.1%	95.3% 4.7%
2	1 0.0%	6541 10.9%	40 0.1%	25 0.0%	13 0.0%	16 0.0%	31 0.1%	33 0.1%	70 0.1%	22 0.0%	96.3% 3.7%
3	24 0.0%	41 0.1%	5430 9.0%	152 0.3%	36 0.1%	40 0.1%	40 0.1%	58 0.1%	66 0.1%	14 0.0%	92.0% 8.0%
4	12 0.0%	31 0.1%	115 0.2%	5509 9.2%	7 0.0%	155 0.3%	0 0.0%	40 0.1%	117 0.2%	63 0.1%	91.1% 8.9%
5	18 0.0%	8 0.0%	58 0.1%	9 0.0%	5475 9.1%	56 0.1%	34 0.1%	65 0.1%	43 0.1%	142 0.2%	92.7% 7.3%
6	44 0.1%	23 0.0%	23 0.0%	202 0.3%	10 0.0%	4824 8.0%	65 0.1%	19 0.0%	158 0.3%	27 0.0%	89.4% 10.6%
7	36 0.1%	6 0.0%	71 0.1%	8 0.0%	51 0.1%	85 0.1%	5687 9.5%	3 0.0%	39 0.1%	5 0.0%	94.9% 5.1%
8	5 0.0%	16 0.0%	52 0.1%	52 0.1%	19 0.0%	30 0.1%	2 0.0%	5857 9.8%	21 0.0%	128 0.2%	94.7% 5.3%
9	26 0.0%	66 0.1%	96 0.2%	106 0.2%	41 0.1%	101 0.2%	14 0.0%	33 0.1%	5234 8.7%	47 0.1%	90.8% 9.2%
10	14 0.0%	10 0.0%	28 0.0%	53 0.1%	170 0.3%	38 0.1%	7 0.0%	144 0.2%	58 0.1%	5467 9.1%	91.3% 8.7%
	97.0% 3.0%	97.0% 3.0%	91.1% 8.9%	89.9% 10.1%	93.7% 6.3%	89.0% 11.0%	96.1% 3.9%	93.5% 6.5%	89.5% 10.5%	91.9% 8.1%	92.9% 7.1%

Figure 3: Confusion matrix with 10 hidden nodes


Confusion Matrix
Output Class
Target Class

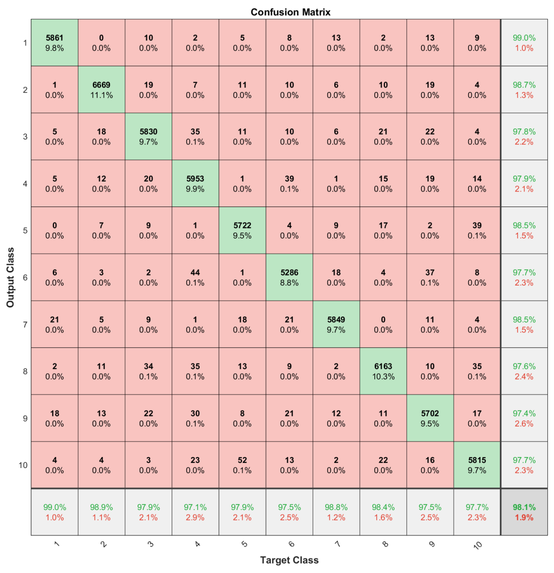
	1	2	3	4	5	6	7	8	9	10	
1	5790 9.7%	1 0.0%	24 0.0%	17 0.0%	11 0.0%	32 0.1%	31 0.1%	17 0.0%	17 0.0%	29 0.0%	97.0% 3.0%
2	5 0.0%	6618 11.0%	19 0.0%	15 0.0%	10 0.0%	18 0.0%	11 0.0%	19 0.0%	72 0.1%	9 0.0%	97.4% 2.6%
3	7 0.0%	22 0.0%	5627 9.4%	80 0.1%	20 0.0%	21 0.0%	19 0.0%	49 0.1%	51 0.1%	9 0.0%	95.3% 4.7%
4	7 0.0%	18 0.0%	46 0.1%	5729 9.5%	8 0.0%	106 0.2%	1 0.0%	15 0.0%	62 0.1%	52 0.1%	94.8% 5.2%
5	14 0.0%	11 0.0%	46 0.1%	4 0.0%	5559 9.3%	17 0.0%	29 0.0%	43 0.1%	17 0.0%	110 0.2%	95.0% 5.0%
6	26 0.0%	11 0.0%	21 0.0%	127 0.2%	4 0.0%	5057 8.4%	57 0.1%	5 0.0%	76 0.1%	37 0.1%	93.3% 6.7%
7	19 0.0%	3 0.0%	38 0.1%	3 0.0%	32 0.1%	62 0.1%	5723 9.5%	6 0.0%	33 0.1%	4 0.0%	96.6% 3.4%
8	5 0.0%	14 0.0%	39 0.1%	53 0.1%	24 0.0%	10 0.0%	5 0.0%	6035 10.1%	24 0.0%	66 0.1%	96.2% 3.8%
9	35 0.1%	38 0.1%	82 0.1%	68 0.1%	30 0.1%	72 0.1%	37 0.1%	16 0.0%	5470 9.1%	38 0.1%	92.9% 7.1%
10	15 0.0%	6 0.0%	16 0.0%	35 0.1%	144 0.2%	26 0.0%	5 0.0%	60 0.1%	29 0.0%	5595 9.3%	94.3% 5.7%
	97.8% 2.2%	98.2% 1.8%	94.4% 5.6%	93.4% 6.6%	95.2% 4.8%	93.3% 6.7%	96.7% 3.3%	96.3% 3.7%	93.5% 6.5%	94.0% 6.0%	95.3% 4.7%

Figure 4: Confusion matrix with 20 hidden nodes


Confusion Matrix
Output Class
Target Class

	1	2	3	4	5	6	7	8	9	10	
1	5831 9.7%	1 0.0%	15 0.0%	5 0.0%	9 0.0%	15 0.0%	21 0.0%	8 0.0%	19 0.0%	8 0.0%	98.3% 1.7%
2	0 0.0%	6630 11.1%	21 0.0%	10 0.0%	8 0.0%	4 0.0%	3 0.0%	13 0.0%	34 0.1%	10 0.0%	98.5% 1.5%
3	9 0.0%	27 0.0%	5757 9.6%	40 0.1%	15 0.0%	10 0.0%	10 0.0%	39 0.1%	19 0.0%	6 0.0%	97.0% 3.0%
4	4 0.0%	10 0.0%	24 0.0%	5891 9.8%	5 0.0%	84 0.1%	3 0.0%	21 0.0%	49 0.1%	26 0.0%	96.3% 3.7%
5	4 0.0%	6 0.0%	25 0.0%	2 0.0%	5658 9.4%	6 0.0%	19 0.0%	25 0.0%	13 0.0%	89 0.1%	96.8% 3.2%
6	15 0.0%	2 0.0%	11 0.0%	57 0.1%	3 0.0%	5199 8.7%	37 0.1%	5 0.0%	45 0.1%	19 0.0%	96.4% 3.6%
7	18 0.0%	4 0.0%	11 0.0%	11 0.0%	19 0.0%	41 0.1%	5805 9.7%	3 0.0%	25 0.0%	3 0.0%	97.7% 2.3%
8	10 0.0%	11 0.0%	37 0.1%	40 0.1%	16 0.0%	10 0.0%	2 0.0%	6085 10.1%	11 0.0%	55 0.1%	96.9% 3.1%
9	24 0.0%	42 0.1%	52 0.1%	58 0.1%	9 0.0%	31 0.1%	17 0.0%	5 0.0%	5606 9.3%	34 0.1%	95.4% 4.6%
10	8 0.0%	9 0.0%	5 0.0%	17 0.0%	100 0.2%	21 0.0%	1 0.0%	61 0.1%	30 0.1%	5699 9.5%	95.8% 4.2%
	98.4% 1.6%	98.3% 1.7%	96.6% 3.4%	96.1% 3.9%	96.9% 3.1%	95.9% 4.1%	98.1% 1.9%	97.1% 2.9%	95.8% 4.2%	95.8% 4.2%	96.9% 3.1%

Figure 5: Confusion matrix with 40 hidden nodes


Confusion Matrix
Output Class
Target Class

	1	2	3	4	5	6	7	8	9	10	
1	5861 9.8%	0 0.0%	10 0.0%	2 0.0%	5 0.0%	8 0.0%	13 0.0%	2 0.0%	13 0.0%	9 0.0%	99.0% 1.0%
2	1 0.0%	6669 11.1%	19 0.0%	7 0.0%	11 0.0%	10 0.0%	6 0.0%	10 0.0%	19 0.0%	4 0.0%	98.7% 1.3%
3	5 0.0%	18 0.0%	5830 9.7%	35 0.1%	11 0.0%	10 0.0%	6 0.0%	21 0.0%	22 0.0%	4 0.0%	97.8% 2.2%
4	5 0.0%	12 0.0%	20 0.0%	5953 9.9%	1 0.0%	39 0.1%	1 0.0%	15 0.0%	19 0.0%	14 0.0%	97.9% 2.1%
5	0 0.0%	7 0.0%	9 0.0%	1 0.0%	5722 9.5%	4 0.0%	9 0.0%	17 0.0%	2 0.0%	39 0.1%	98.5% 1.5%
6	6 0.0%	3 0.0%	2 0.0%	44 0.1%	1 0.0%	5286 8.8%	18 0.0%	4 0.0%	37 0.1%	8 0.0%	97.7% 2.3%
7	21 0.0%	5 0.0%	9 0.0%	1 0.0%	18 0.0%	21 0.0%	5849 9.7%	0 0.0%	11 0.0%	4 0.0%	98.5% 1.5%
8	2 0.0%	11 0.0%	34 0.1%	35 0.1%	13 0.0%	9 0.0%	2 0.0%	6163 10.3%	10 0.0%	35 0.1%	97.6% 2.4%
9	18 0.0%	13 0.0%	22 0.0%	30 0.1%	8 0.0%	21 0.0%	12 0.0%	11 0.0%	5702 9.5%	17 0.0%	97.4% 2.6%
10	4 0.0%	4 0.0%	3 0.0%	23 0.0%	52 0.1%	13 0.0%	2 0.0%	22 0.0%	16 0.0%	5815 9.7%	97.7% 2.3%
	99.0% 1.0%	98.9% 1.1%	97.9% 2.1%	97.1% 2.9%	97.9% 2.1%	97.5% 2.5%	98.8% 1.2%	98.4% 1.6%	97.5% 2.5%	97.7% 2.3%	98.1% 1.9%

Figure 6: Confusion matrix with 100 hidden nodes

Performance Curves

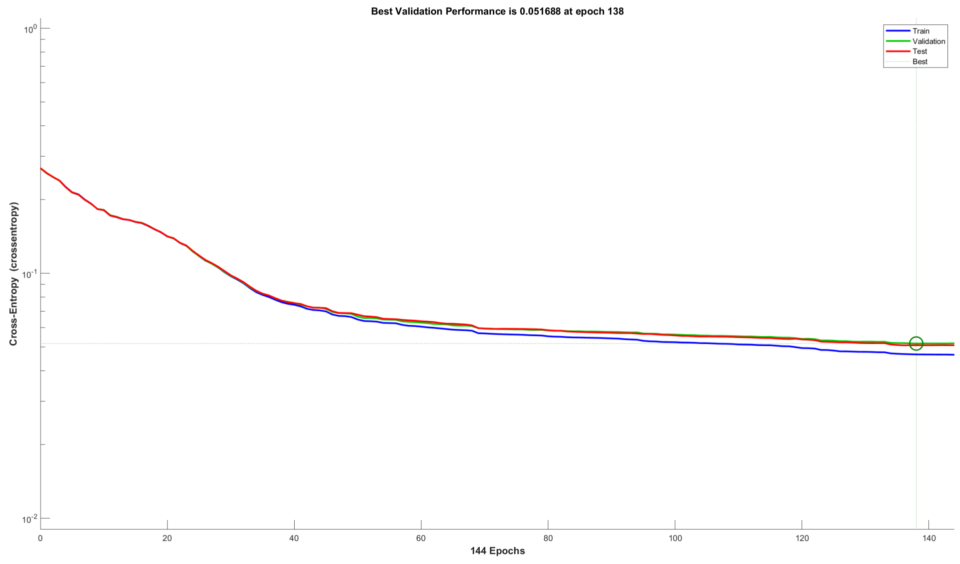


Figure 7: Performance curve with 5 hidden nodes

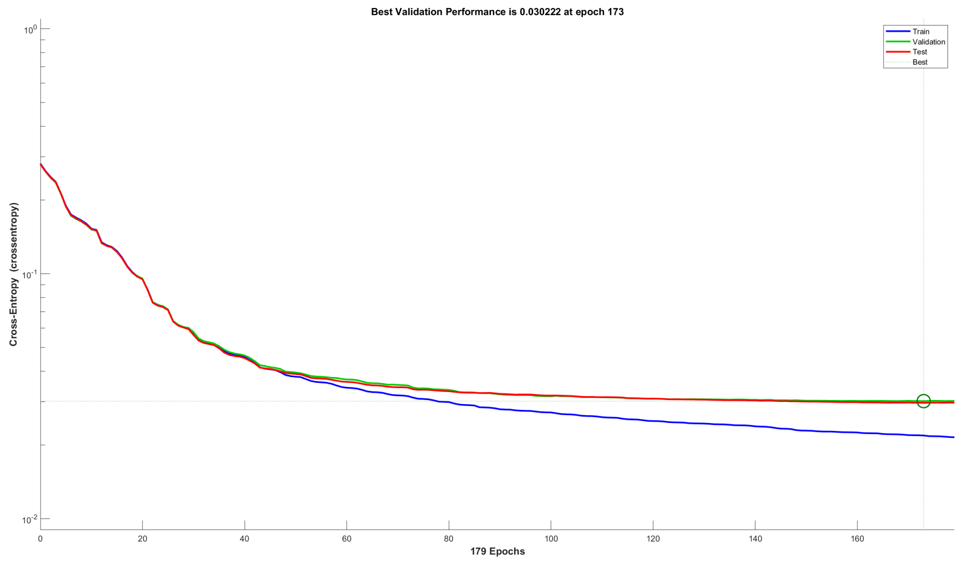


Figure 8: Performance curve with 10 hidden nodes

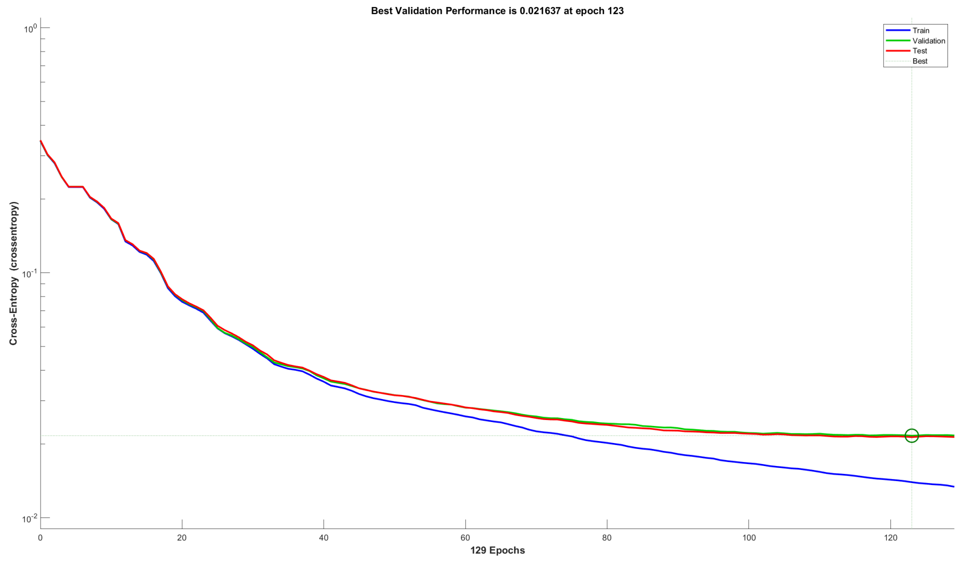


Figure 9: Performance curve with 20 hidden nodes

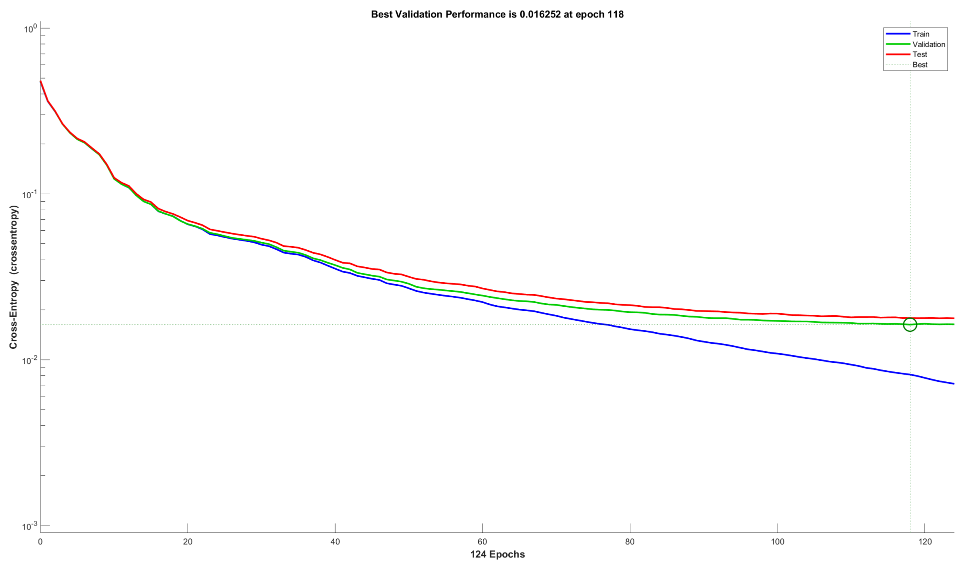


Figure 10: Performance curve with 40 hidden nodes

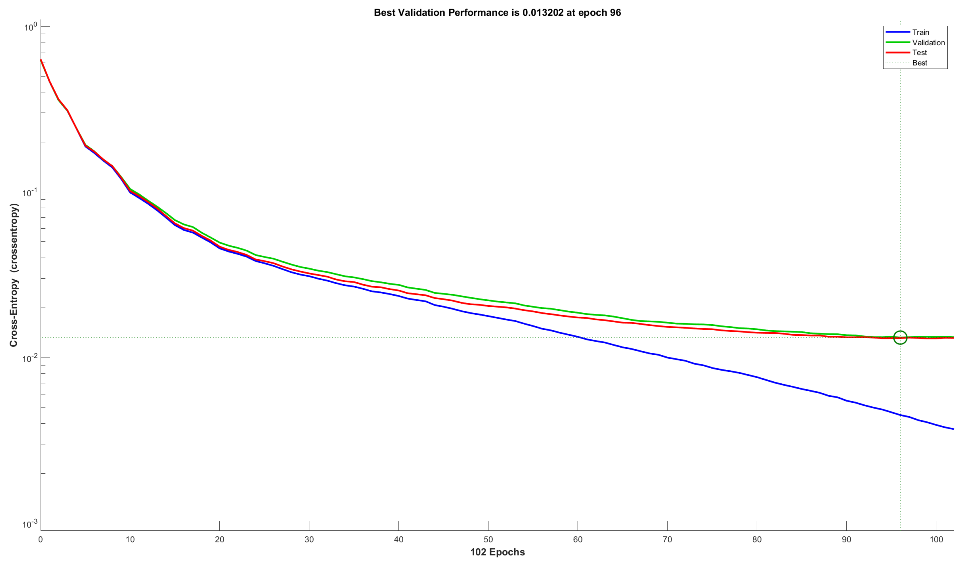


Figure 11: Performance curve with 100 hidden nodes

Plots of Rate of Correctness (ROC)

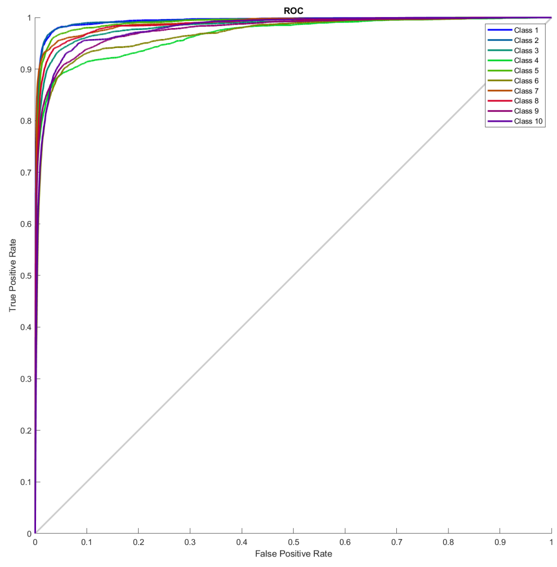


Figure 12: ROC curve with 5 hidden nodes

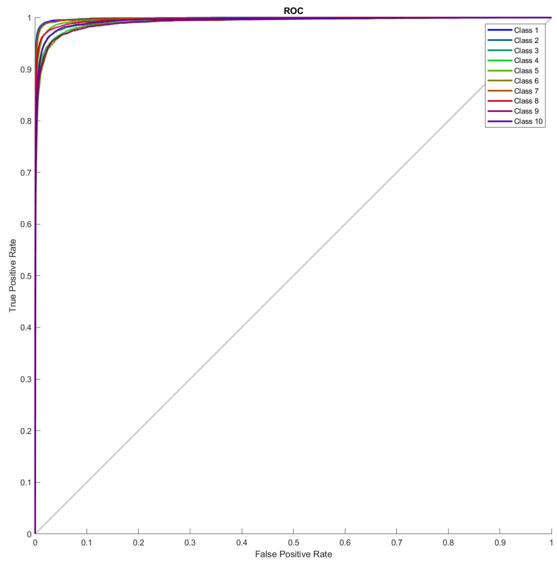


Figure 13: ROC curve with 10 hidden nodes

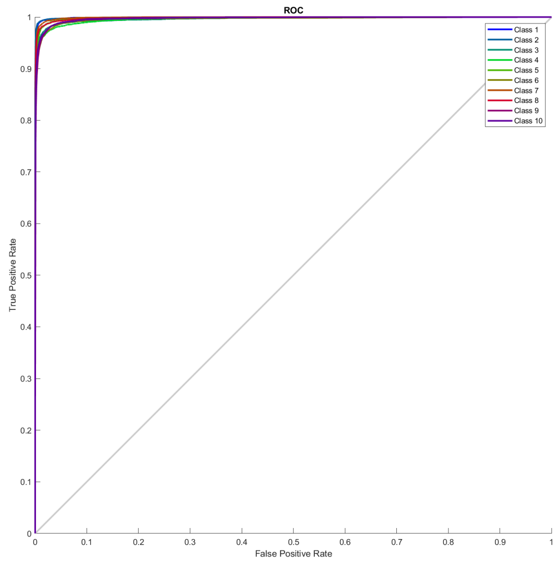


Figure 14: ROC curve with 20 hidden nodes

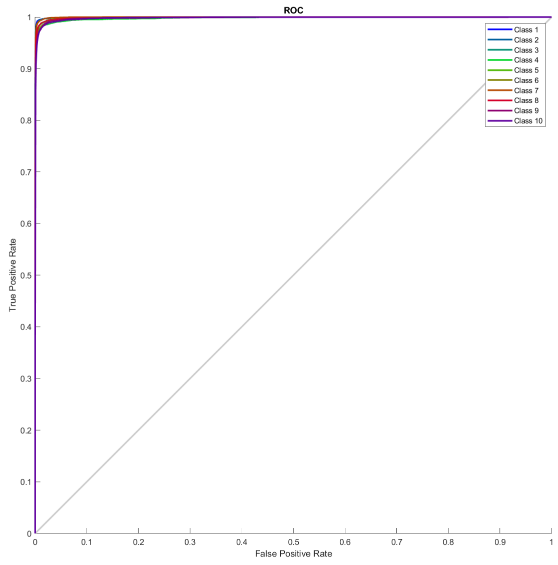


Figure 15: ROC curve with 40 hidden nodes

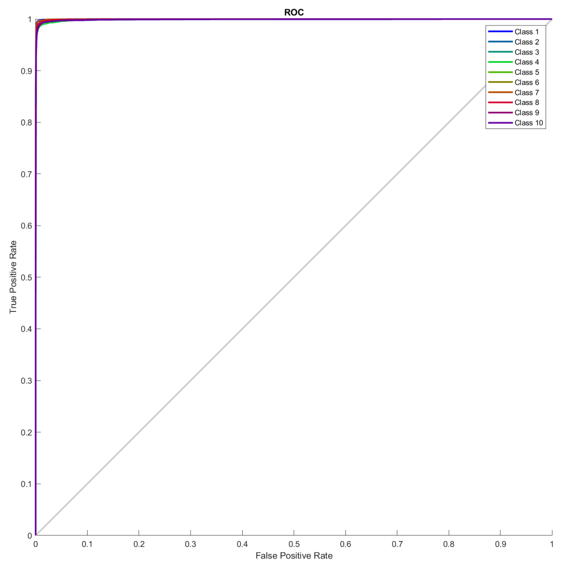


Figure 16: ROC curve with 100 hidden nodes

Conclusion

- **Confusion matrix:** With more hidden nodes, the confusion rate drops accordingly. See the table below.

Hidden nodes	Total confusion rate
5	11.5%
10	8.9%
20	5.6%
40	2.9%
100	1.9%

Table 1: Confusion rates

- **Performance:** With more hidden nodes, the error rates drop at both validation set and testing set. Meanwhile, it takes a longer time before it stops and converges, in spite of fewer iterations. Another interpretation: The network has better capability of fitting the pattern with more hidden nodes, and therefore the training process is smoother and we need fewer iterations in training. But it takes more time since we have much more parameters to train, which leads to a longer time for each iteration step.

- **ROC curve:** With more hidden nodes, the ROC curve is closer to the left top corner, which indicates a better performance of classification.

Hidden nodes	Total training epochs
5	500
10	185
20	125
40	125
100	110

Table 2: Training iterations

Max Confusion

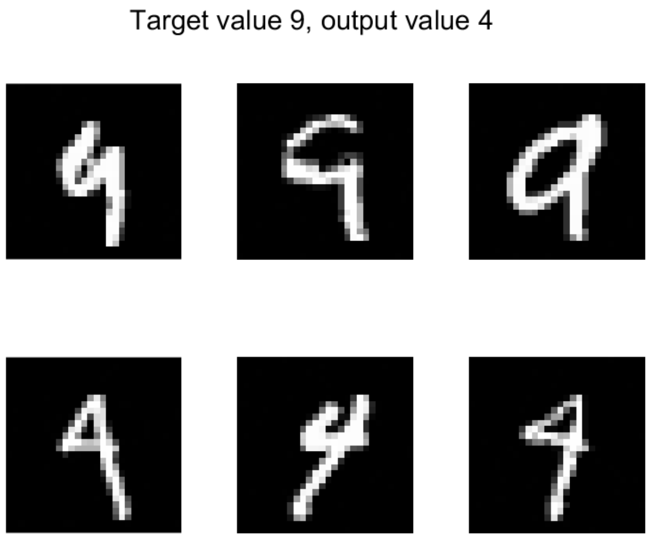


Figure 17: Max confusion at target value “4”, which is mistakenly output as “9”

Question 2.1

```
target_size = [patchsize, patchsize, 1];
source_size = size(IMAGES);
indices = [randi(source_size(1) - target_size(1) + 1, [1, numpatches]); ...
    randi(source_size(2) - target_size(2) + 1, [1, numpatches]); ...
    randi(source_size(3) - target_size(3) + 1, [1, numpatches])];
for i = 1 : numpatches
    t = indices(:, i)';
    temp = IMAGES(t(1) : (t(1) + target_size(1) - 1), ...
        t(2) : (t(2) + target_size(2) - 1), ...
        t(3) : (t(3) + target_size(3) - 1));
    patches(:, i) = reshape(temp, [numel(temp), 1]);
end
```

Question 2.2

The sizes of W_1, w_1, W_2 and w_2 are, respectively, 25×64, 25×1, 64×25 and 64×1. Every line of code is vectorized.

```
N = size(data, 2);
x_0 = data;                    % 64-by-N

z_1 = W1 * data + w1;          % 25-by-N
x_1 = sigmoid(z_1);            % 25-by-N

z_2 = W2 * x_1 + w2;           % 64-by-N
x_2 = sigmoid(z_2);            % 64-by-N

% Forward propagation

J = 1/2 * sumsqr(x_2 - x_0) / N;
R = lambda/2 * (sumsqr(W1) + sumsqr(W2));
rho_hat = mean(x_1, 2);        % 25-by-1
rho = sparsityParam;
S = beta * sum( rho*log(rho./rho_hat) + (1-rho)*log((1-rho)./(1-rho_hat)) );

cost = J + R + S;

% Back propagation

delta_2 = (x_2 - x_0) .* x_2 .* (1 - x_2);                             % 64-by-N

W2grad = (delta_2 * x_1') / N + lambda * W2;                           % 64-by-25
w2grad = mean(delta_2, 2);                                             % 64-by-1

rhograd_2 = beta * ( - rho./rho_hat + (1-rho)./(1-rho_hat) );

rhograd_1 = (x_1 .* (1 - x_1)) * x_0' / N;

rhograd_0 = mean(x_1 .* (1 - x_1), 2);

delta_1 = (W2' * delta_2) .* x_1 .* (1 - x_1);                         % 25-by-N

W1grad = (delta_1 * x_0') / N + lambda * W1 + rhograd_2 .* rhograd_1; % 25-by-64
w1grad = mean(delta_1, 2) + rhograd_2 .* rhograd_0;                    % 25-by-1
```

Question 2.3

(1)

Note that S does not depend on z_i^2, we obtain

$$\delta_i^2 = \frac{\partial}{\partial z_i^2}\left[\frac{1}{2}\sum_{j=1}^{n}(f(z_j^2) - x_j)^2 + S\right]$$

$$= (f(z_i^2) - x_i)\frac{\partial}{\partial z_i^2} f(z_i^2)$$

$$= (f(z_i^2) - x_i) \cdot f(z_i^2)(1 - f(z_i^2))$$

$$= (\hat{x}_i - x_i) \cdot \hat{x}_i(1 - \hat{x}_i)$$

where

$$f(z_i^2) = x_i^2 = \hat{x}_i = \frac{1}{1 + \exp(-z_i^2)} = \left[1 + \exp\left(-w_0^2 - \sum_{j=1}^{n_1} W_{ji}^2 x_j^1\right)\right]^{-1}.$$

(2)

By definition, the simple gradient descent algorithm uses the iterations below:

$$W_{ij}^1 \leftarrow W_{ij}^1 - \alpha \frac{\partial J}{\partial W_{ij}^1},$$

$$w_0^2 \leftarrow w_0^2 - \alpha \frac{\partial J}{\partial w_0^2},$$

where α is the learning rate with the gradients

$$\frac{\partial J}{\partial W_{ij}^1} = \frac{\partial J_s}{\partial z_j^1}\frac{\partial z_j^1}{\partial W_{ij}^1} + \frac{\partial R}{\partial W_{ij}^1} = d_j^1 x_i + \lambda W_{ij}^1,$$

$$\frac{\partial J}{\partial w_0^2} = \frac{\partial J_s}{\partial z_j^2}\frac{\partial z_j^2}{\partial w_0^2} + \frac{\partial R}{\partial w_0^2} = d_j^2.$$

(3)

By chain rule of multivariate derivatives, we obtain

$$\delta_i^1 = \frac{\partial J}{\partial z_i^1} = \frac{\partial J}{\partial x_i^1}\frac{\partial x_i^1}{\partial z_i^1}$$

$$= \frac{\partial J}{\partial x_i^1} \cdot x_i^1(1 - x_i^1)$$

$$= \left[\frac{\partial J_s}{\partial x_i^1} + \frac{\partial S}{\partial x_i^1}\right] \cdot x_i^1(1 - x_i^1)$$

$$= \left[\sum_{j=1}^{n_1} \frac{\partial J_s}{\partial z_j^2}\frac{\partial z_j^2}{\partial x_i^1} + \frac{\partial S}{\partial x_i^1}\right] \cdot x_i^1(1 - x_i^1)$$

$$= \left[\sum_{j=1}^{n_1} \delta_j^2 W_{ij}^2 + \frac{\partial S}{\partial x_i^1}\right] \cdot x_i^1(1 - x_i^1),$$

where

$$\frac{\partial S}{\partial x_i^1} = \frac{\partial S}{\partial \hat{\rho}_i} = \beta\left(-\frac{\rho}{\hat{\rho}_i} + \frac{1 - \rho}{1 - \hat{\rho}_i}\right).$$

Question 2.4

Given the hint, it is acceptable not to use vectorized manipulations here.

```
eps = 1e-6;
for i = 1 : length(theta)
    dtheta = zeros(size(theta));
    dtheta(i) = eps;
    numgrad(i) = (J(theta + dtheta) - J(theta - dtheta)) / (2*eps);
end
```

Run “Step 3” in “train.m”, and the output indicates that the numerical and the analytical gradients correspond. The error is about 10^{-11}.

```
38.0000    38.0000
12.0000    12.0000

The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

2.1452e-12

Norm of the difference between numerical and analytical gradient (should be < 1e-9)

8.2413e-11
```

Question 2.5

Thanks to the vectorized expressions, the training process takes only 30 seconds before it stops at 400 iterations. The final cost is 0.4440 with all paramters at default value.

Question 2.6

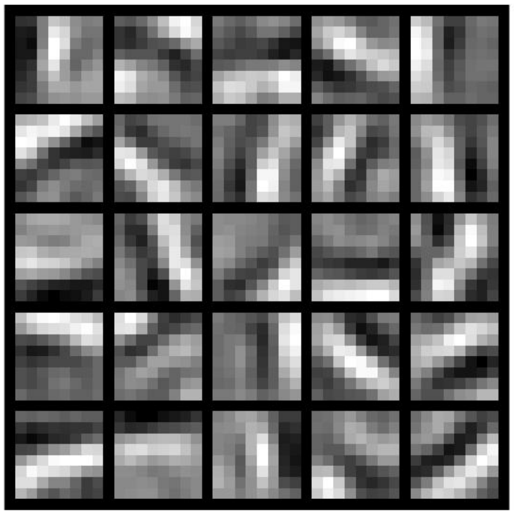


Figure 18: Visualization of the weights

Source Code

Please download the souece code from http://39.106.23.58/files/PR4_2015011506.7z

For Question 1, please “cd ./src1” and run “main.m”. It may take minutes to train the network, but the result is reproducible because of the random seed.

For Question 2, please “cd ./src2” and run “train.m” by section.