Pattern recognition with neural networks (OM/GCED)

Authors: *Alex Carrillo Alza, Oriol Narvaez Serrano* (2nd, GCED)

Contents

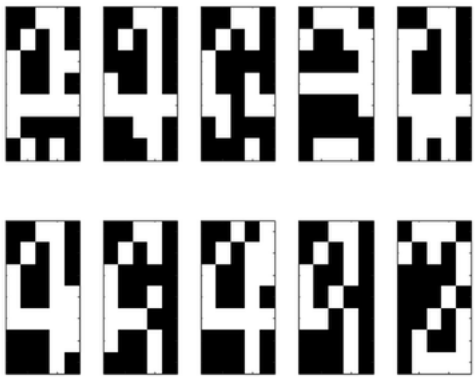
1. Generation of the training data set

Let `Xtr,ytr` be the training data set.

```
num_target = [4];
noise_freq = 0.1;
tr_freq    = 0.5;       % Training parameters.
tr_p       = 500;
tr_seed    = 123456;

% Training data generation.
[Xtr, ytr] = om_uo_nn_dataset(tr_seed, tr_p, num_target, tr_freq, noise_freq);
```

Preview of some training data generated `Xtr`.

```
for i=1:10
    subplot(2,5,i); numplot(Xtr,i);
end
```



2. Definition of the loss function and its gradient

If we define the sigmoid matrix of inputs $\sigma(X^{TR})$ and the row vector of residuals $y(X^{TR},w)$ as

```
sig = @(X)   1./(1+exp(-X));    % Sigmoid matrix.
y   = @(X,w) sig(w'*sig(X));    % Row vector of residuals.
```

Then, the value of the loss function $\tilde{L}$ and its gradient $\nabla\tilde{L}$ can be expressed as

```
la = 0.0;                                                          % Lambda.
L  = @(w) norm(y(Xtr,w)-ytr)^2 + (la*norm(w)^2)/2;                 % Loss function.
gL = @(w) 2*sig(Xtr)*((y(Xtr,w)-ytr).*y(Xtr,w).*(1-y(Xtr,w)))'+la*w;    % Gradient.
```

3. Finding of the value *w** minimizing *L(w)*

Now, the value of w^* minimizing $\tilde{L}(w;X^{TR},y^{TR},\lambda)$ can be found using the optimization routines developed during the course.

```
w = zeros(1, 35)';  % Weights of the SLNN.
h = []; Q = [];     % Unused hessian forms.

kmax    = 1000;  epsG   = 1.0e-06;
kmaxBLS = 30;    epsBLS = 1.0e-03;   % Parameters of methods.

almax = 2; almin = 10^-3; rho = 0.5; c1 = 0.01; c2 = 0.45;
iW = 2; isd = 1; icg = 1; irc = 1; nu = 0.1; delta = 0.1;
```

```
% Optimization with the chosen method.
[xk, dk, alk, iWk, betak] = DM(w,L,gL,h,Q,epsG,kmax,almax,almin,rho,c1,c2,iW,isd,icg,irc,nu,delta,kmaxBLS,epsBLS);
```

4. Computation of the training accuracy

The training accuracy is computated using the formula $\frac{100}{p} \cdot \sum_{j=1}^{p} \delta[y(x_j^{TR}, w^*)], y^{TR}$

```
iWk = [iWk, NaN];
niter = size(xk,2); xo = xk(:,niter);     % Number of iterations performed.

tr_accuracy = 0;
train = round(y(Xtr, xk(:,niter)));       % Prediction of training data.
for j = 1:tr_p
    if train(j) == ytr(j) tr_accuracy = tr_accuracy + 1; end
end
tr_accuracy = 100/tr_p * tr_accuracy;     % Percentage of correct answers.
```

5. Generation of the test data set

Let `Xte,yte` be the test data set.

```
te_q    = 5000;
te_seed = 789101;   % Test parameters.

% Test data generation.
[Xte, yte] = om_uo_nn_dataset(te_seed, te_q, num_target, tr_freq, noise_freq);
```

6. Computation of the test accuracy

The test accuracy is computated using the formula $\frac{100}{q} \cdot \sum_{j=1}^{q} \delta[y(x_j^{TE}, w^*)], y^{TE}$

```
te_accuracy = 0;
test = round(y(Xte, xk(:,niter)));        % Prediction of training data.
for j = 1:te_q
    if test(j) == yte(j) te_accuracy = te_accuracy + 1; end
end
te_accuracy = 100/te_q * te_accuracy;     % Percentage of correct answers.
```

7. Computation of some analysis information

The objective function and gradient are evaluated for every point. Also the rate of convergence and the descent condition is calculated for every iteration.

```
fk = []; gk = []; rk = [NaN]; gdk = [];
for k = 1:niter fk = [fk,L(xk(:,k))]; gk=[gk,gL(xk(:,k))]; end    % f(xk) and g(xk).
for k = 2:niter rk = [rk,(fk(k)-L(xo))/(fk(k-1)-L(xo))]; end      % Rate of convergence.
for k = 1:niter-1 gdk = [gdk,gk(:,k)'*dk(:,k)]; end               % Descent condition.
gdk = [gdk,0];
```

8. Visualization of the results

These are the results of the SLNN for the given target.

```
fprintf('::::::::::::::::::::::::::::::::::::::::::::::::::: \n');
fprintf(' Pattern recognition with neural networks (OM/GCED) \n');
fprintf('::::::::::::::::::::::::::::::::::::::::::::::::::: \n');
fprintf('Training data set generation. \n');
fprintf('    num_target = %d \n', num_target);
fprintf('    noise_freq = %.2f \n', noise_freq);
fprintf('    tr_freq    = %.2f \n', tr_freq);
fprintf('    tr_p       = %d \n', tr_p);
fprintf('    tr_seed    = %d \n', tr_seed);
fprintf('Optimization \n');
fprintf('    L2 reg. lambda = %.2f \n', la);
fprintf('    epsG    = %.1e,  kmax = %d \n', epsG, kmax);
fprintf('    kmaxBLS = %d,  epsBLS = %.1e \n', kmaxBLS, epsBLS);
fprintf('    c1 = %.2f,  c2 = %.2f,  isd = %d \n', c1, c2, isd);
fprintf(' - Algorithm: \n');
fprintf('    k     alpha   wolfe     g''*d      f(w)     ||g||     r \n');
fprintf('%5d %9.1e %5d %10.1e %9.1e %9.1e %9.1e \n', 1, alk(1), iWk(1), gdk(1), L(xk(1)), norm(gk(:,1)), rk(1));
fprintf('%5d %9.1e %5d %10.1e %9.1e %9.1e %9.1e \n', 2, alk(2), iWk(2), gdk(2), L(xk(2)), norm(gk(:,2)), rk(2));
fprintf('   ...     ...     ...       ...       ...       ...     ... \n');
fprintf('%5d %9.1e %5d %10.1e %9.1e %9.1e %9.1e \n', niter-1, alk(niter-1), iWk(niter-1), gdk(niter-2), L(xk(niter-1)), norm(gk(:,niter-1)), rk(ni
fprintf('%5d %9.1e %5d %10.1e %9.1e %9.1e %9.1e \n', niter, alk(niter), iWk(niter), gdk(niter-1), L(xk(niter)), norm(gk(:,niter)), rk(niter));
fprintf('w* = [ \n       ');
for i = 1:length(w)
    fprintf('%8.1e,', xk(i,niter)); if mod(i,5) == 0 fprintf('\n       '); end
end
fprintf('                                          ] \n');
fprintf('Test data set generation. \n');
fprintf('    te_q    = %d \n', te_q);
fprintf('    te_seed = %d \n', te_seed);
fprintf('Accuracy. \n');
fprintf('    tr_accuracy = %.1f \n', tr_accuracy);
fprintf('    te_accuracy = %.1f \n', te_accuracy);
fprintf('\n');
```

9. Assignment

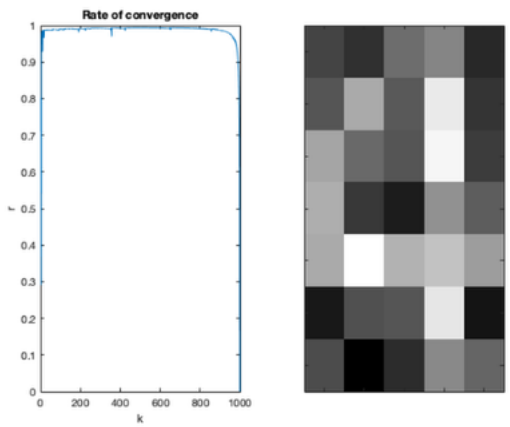
The task to be done are:

a) Solve the pattern recognition problem for `num_target=[4]` and `num_target=[8]` (separately) with all the optimization methods and variants studied in this course. Take `kmax=1000` and `lambda=0.0`, `tr_freq=0.5` and `noise_freq=0.1`. Report the behavior observed for each method with a table similar to that on slide #20. Analize the convergence of each method and the value of `Accuracy_TR` and `Accuracy_TE`.

TARGET = [4]:

Gradient Method

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 4
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 1
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -1.7e+03   4.4e+03   4.1e+01       NaN
    2   1.2e-01     2   -6.5e+04   4.4e+03   2.5e+02   8.0e-01
   ...     ...     ...       ...       ...       ...     ...
  999   4.9e+00     0   -1.6e-05   4.5e+03   3.2e-03   5.0e-01
 1000   7.5e+00   NaN   -1.0e-05   4.5e+03   3.9e-03   0.0e+00
w* = [
       -2.9e+00,-4.3e+00,-5.6e-01, 9.6e-01,-4.7e+00,
       -1.9e+00, 3.3e+00,-1.7e+00, 7.2e+00,-3.9e+00,
        3.0e+00,-8.1e-01,-1.8e+00, 8.0e+00,-3.3e+00,
        3.5e+00,-3.6e+00,-5.5e+00, 1.7e+00,-1.4e+00,
        3.4e+00, 8.6e+00, 3.8e+00, 4.9e+00, 2.6e+00,
       -5.6e+00,-2.2e+00,-1.9e+00, 7.1e+00,-5.9e+00,
       -2.5e+00,-7.3e+00,-4.4e+00, 1.2e+00,-8.9e-01,
                                                    ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 100.0
```

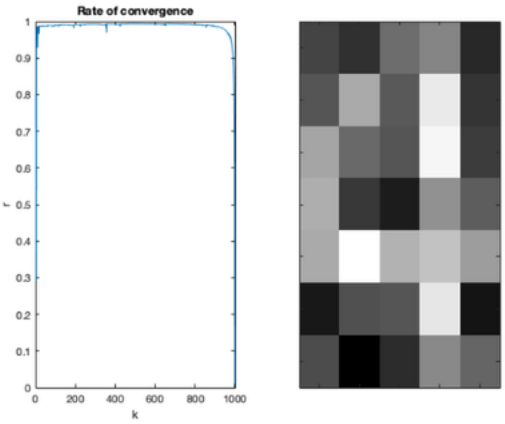


As we can see, the target 4 is an easy number to detect because it gets a very good accuracy in both training and test. Specifically, using the Gradient Method, although it spends the maximum iterations it gets a perfect result of 100% in both accuracies. We can not say that the method converges to the global minimum but the point obtained is close enough to get a workable solution to the problem.

Conjugate Gradient Method

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 4
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 2
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0    3.1e+02   4.4e+03   4.1e+01       NaN
```

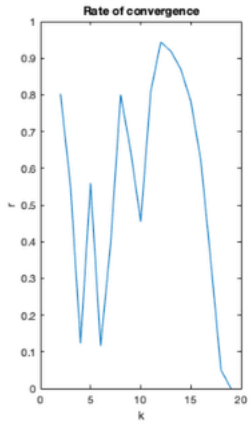
```
    2   1.2e-01     2   -5.1e+04   4.4e+03   2.5e+02   8.0e-01
   ...     ...     ...        ...        ...       ...     ...
 1000   7.5e+00     0    3.5e-08   4.5e+03   3.9e-03   5.0e-01
 1001   4.9e+00   NaN   -4.8e-09   4.9e+03   3.1e-03   0.0e+00
w* = [
       -2.9e+00,-4.3e+00,-5.6e-01, 9.5e-01,-4.7e+00,
       -1.9e+00, 3.3e+00,-1.7e+00, 7.2e+00,-3.9e+00,
        3.0e+00,-8.1e-01,-1.8e+00, 8.0e+00,-3.3e+00,
        3.5e+00,-3.6e+00,-5.5e+00, 1.7e+00,-1.4e+00,
        3.4e+00, 8.6e+00, 3.8e+00, 4.9e+00, 2.6e+00,
       -5.6e+00,-2.2e+00,-1.9e+00, 7.1e+00,-5.9e+00,
       -2.5e+00,-7.3e+00,-4.4e+00, 1.2e+00,-8.9e-01,
                                                     ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 100.0
```

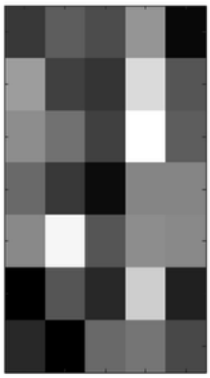


With the Conjugate Gradient Method, we get a similar result. The method also takes the maximum iterations (plus 1, because it starts at 1, not at 0) and it gets a perfect result of 100% in both training and test accuracies. It should be noted that it reaches the same point solution as the Gradient Method, so they are quite similar in convergence.

Quasi-Newton Method: BFGS

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 4
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 3
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -1.7e+03   4.4e+03   4.1e+01       NaN
    2   1.2e-01     2   -4.6e+04   4.4e+03   2.5e+02   8.0e-01
   ...     ...     ...        ...        ...       ...     ...
   18   1.1e+01     0   -1.5e-03   4.4e+03   1.8e-04   5.0e-02
   19   5.2e+02   NaN   -1.5e-05   4.4e+03   3.6e-10   0.0e+00
w* = [
       -9.9e+01,-2.3e+01,-5.8e+01, 9.6e+01,-2.0e+02,
        1.1e+02,-8.2e+01,-1.0e+02, 2.3e+02,-3.6e+01,
        7.8e+01, 2.0e+01,-7.5e+01, 3.1e+02,-2.3e+01,
        6.3e+00,-9.2e+01,-1.9e+02, 5.9e+01, 6.2e+01,
        6.9e+01, 2.9e+02,-3.7e+01, 8.2e+01, 6.9e+01,
       -2.2e+02,-4.2e+01,-1.3e+02, 2.1e+02,-1.4e+02,
       -1.3e+02,-2.1e+02, 7.3e+00, 2.5e+01,-6.0e+01,
                                                     ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 99.7
```



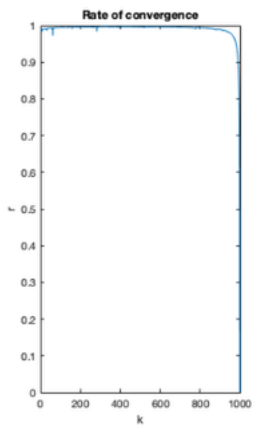


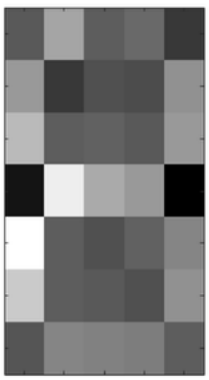
The BFGS is above all the fastest method. It only takes 19 iterations to reach a workable solution. However, it doesn't get a perfect result, instead it obtains a 99'7% test accuracy, which still is very good as the convergence fluctuates.

TARGET = [8]:

Gradient Method

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 8
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 1
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -8.8e+02   4.4e+03   3.0e+01       NaN
    2   2.0e-03     0   -5.9e+02   4.4e+03   2.4e+01   9.9e-01
   ...     ...     ...       ...       ...       ...     ...
  999   5.9e-03     0   -3.4e+00   4.4e+03   1.7e+00   5.0e-01
 1000   6.6e-03   NaN   -3.0e+00   4.4e+03   1.8e+00   0.0e+00
w* = [
       -1.6e+00, 2.1e+00,-1.2e+00,-7.9e-01,-3.0e+00,
        1.6e+00,-3.0e+00,-1.8e+00,-2.1e+00, 1.2e+00,
        3.1e+00,-1.2e+00,-1.2e+00,-1.6e+00, 1.6e+00,
       -4.7e+00, 5.6e+00, 2.4e+00, 1.6e+00,-5.8e+00,
        6.5e+00,-1.3e+00,-1.9e+00,-1.2e+00, 5.5e-01,
        3.9e+00,-1.4e+00,-1.5e+00,-1.8e+00, 1.2e+00,
       -1.6e+00, 6.4e-01, 4.9e-01, 1.6e-01,-1.2e+00,
                                                    ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 95.6
    te_accuracy = 96.1
```

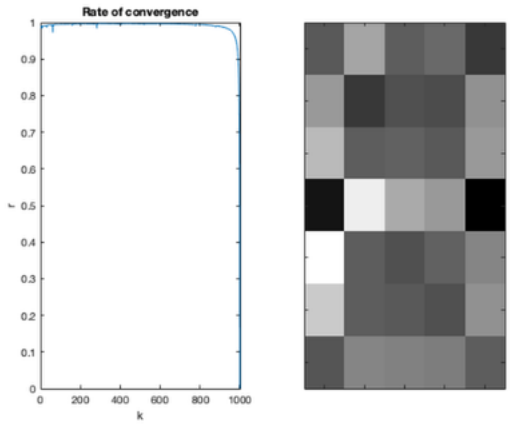




In the case of the target number 8, it is concluded that it is a bit more difficult to detect. As before, the gradient method takes the maximum iterations but it obtains a decrease on the accuracies. As a result, the test accuracy is smaller with a 96'1%, but it is still a good model.

Conjugate Gradient Method

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 8
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 2
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -1.9e+02   4.4e+03   3.0e+01       NaN
    2   2.0e-03     0   -1.5e+02   4.4e+03   2.4e+01   9.9e-01
   ...     ...     ...        ...       ...       ...     ...
 1000   6.6e-03     0    8.3e-03   4.4e+03   1.8e+00   5.0e-01
 1001   5.9e-03   NaN   -3.2e-03   4.5e+03   1.7e+00   0.0e+00
w* = [
       -1.6e+00, 2.1e+00,-1.2e+00,-7.9e-01,-3.0e+00,
        1.6e+00,-3.0e+00,-1.8e+00,-2.1e+00, 1.2e+00,
        3.1e+00,-1.2e+00,-1.2e+00,-1.6e+00, 1.6e+00,
       -4.7e+00, 5.6e+00, 2.4e+00, 1.6e+00,-5.8e+00,
        6.5e+00,-1.3e+00,-1.9e+00,-1.2e+00, 5.5e-01,
        3.9e+00,-1.4e+00,-1.5e+00,-1.8e+00, 1.2e+00,
       -1.6e+00, 6.4e-01, 4.9e-01, 1.6e-01,-1.2e+00,
                                                    ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 95.8
    te_accuracy = 96.2
```

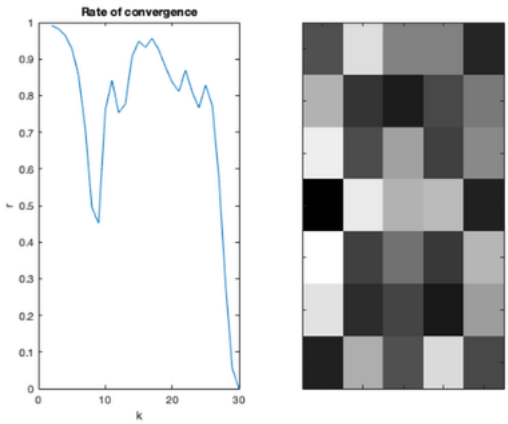


With the Conjugate Gradient Method, we get a similar performance. It takes the maximum iterations and the accuracies are the same as the previous method with a negligible difference. The solution point reached is also the same.

Quasi-Newton Method: BFGS

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 8
    noise_freq = 0.10
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 3
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -8.8e+02   4.4e+03   3.0e+01       NaN
    2   2.0e-03     2   -8.9e+02   4.4e+03   2.4e+01   9.9e-01
   ...     ...     ...        ...       ...       ...     ...
   29   1.7e+01     0   -3.7e-02   4.4e+03   7.1e-03   5.5e-02
   30   5.4e+02   NaN   -4.4e-04   4.4e+03   2.6e-12   0.0e+00
w* = [
       -3.8e+02, 8.4e+02, 4.4e+01, 3.6e+01,-7.7e+02,
        4.5e+02,-6.3e+02,-8.4e+02,-4.6e+02,-2.2e+01,
        9.6e+02,-4.3e+02, 3.3e+02,-5.0e+02, 1.1e+02,
       -1.1e+03, 9.2e+02, 4.4e+02, 5.0e+02,-8.0e+02,
```

```
        1.1e+03,-5.2e+02,-1.0e+02,-5.8e+02, 4.9e+02,
        8.7e+02,-7.0e+02,-4.7e+02,-8.7e+02, 2.8e+02,
       -7.8e+02, 4.3e+02,-3.6e+02, 7.9e+02,-4.3e+02,
                                                      ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 98.8
    te_accuracy = 95.4
```



As in the previous section, the BFGS Method is the fastest. It only takes 30 iterations to reach a workable solution. But it causes the same behaviour as before: the accuracies decrease (in a very small percentage). We still get a very good preformance with a 96,1% of test accuracy.

b) Solve the pattern recognition problem for every digit from 0 to 9 with the BFGS method, `kmax=1000` and `lambda=0.0`, `tr_freq=0.5` and `noise_freq=0.2`. Report the results in a table similar to that on slide #20. Compare the `Accuracy_TE` for every digit and try to find an explanation for the observed differences.

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Pattern recognition with neural networks (OM/GCED)
::::::::::::::::::::::::::::::::::::::::::::::::::::::
Training data set generation.
    num_target = 1
    noise_freq = 0.20
    tr_freq    = 0.50
    tr_p       = 500
    tr_seed    = 123456
Optimization
    L2 reg. lambda = 0.00
    epsG    = 1.0e-06,  kmax = 1000
    kmaxBLS = 30,  epsBLS = 1.0e-03
    c1 = 0.01,  c2 = 0.45,  isd = 3
 - Algorithm:
    k     alpha   wolfe     g'*d      f(w)      ||g||     r
    1   2.0e+00     0   -2.3e+03   4.4e+03   4.8e+01       NaN
    2   2.0e-03     2   -1.5e+03   4.4e+03   3.6e+01   9.8e-01
   ...     ...     ...        ...       ...       ...     ...
   21   5.5e+00     0   -8.3e-04   4.4e+03   1.0e-04   5.7e-02
   22   2.3e+02   NaN   -1.2e-05   4.4e+03   7.3e-16   0.0e+00
w* = [
        1.5e+01,-5.0e+01, 4.3e+01,-7.9e+01,-1.7e+01,
       -2.1e+01, 8.4e+01, 8.5e+01,-3.5e+01,-3.8e+00,
       -5.5e+01,-1.8e+01, 7.8e+01,-3.9e+01,-4.3e+01,
       -2.2e+01,-2.6e+01, 6.8e+01,-7.0e+01, 1.0e+01,
       -1.4e+01,-4.4e+01, 7.9e+01,-3.9e+01,-3.0e+01,
       -5.2e+01,-7.2e+00, 1.2e+02, 7.1e+00,-7.7e+01,
       -3.8e+01, 6.2e+01, 3.4e+01, 3.1e+01,-2.7e+01,
                                                      ]
Test data set generation.
    te_q    = 5000
    te_seed = 789101
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 99.8
```

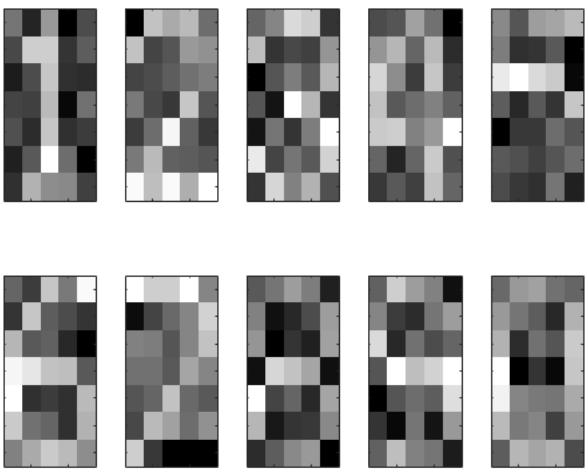
As we can see, the training accuracy with the target number 1 is 100%, perfect scoring and the test accuracy is 99.8%, it identifies the number 1 on almost all the correct places.

Now we are going to do the same with all the other numbers, we are only going to show the final results to not repeat the code.

```
subplot(2,5,1);
numplot(xk(:,niter),1);
for i=2:10
    [TrainingAccuracy, TestAccuracy, Wk] = BFGS_training(i, TrainingAccuracy, TestAccuracy, Wk);
    subplot(2,5,i);
    numplot(Wk(:,end),1)
end
```

```
::::::::::::::::::::
 TARGET NUMBER 2
::::::::::::::::::::
w* = [
       -2.1e+03, 8.8e+02, 5.2e+02, 7.3e+02,-3.8e+02,
        8.9e+02,-9.9e+02,-7.6e+02, 2.8e+02, 1.4e+02,
       -1.0e+03,-8.8e+02,-6.6e+02,-3.4e+02,-1.8e+02,
       -2.4e+02,-9.5e+02,-1.2e+03, 9.2e+02,-7.4e+02,
       -1.1e+03,-4.2e+02, 1.6e+03,-5.5e+02,-1.2e+03,
       -1.9e+02, 7.5e+02,-5.6e+02,-6.2e+02,-7.6e+02,
        1.7e+03, 8.4e+02, 1.7e+03, 5.8e+02, 1.8e+03,
                                                      ]
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 97.6

::::::::::::::::::::
 TARGET NUMBER 3
::::::::::::::::::::
w* = [
       -7.9e+02, 2.1e+02, 2.8e+03, 2.4e+03,-2.3e+03,
        2.0e+03,-2.4e+03,-1.7e+03,-2.0e+03, 6.3e+02,
       -4.0e+03,-1.3e+03,-4.0e+01,-1.2e+03, 1.7e+03,
       -1.4e+03,-3.4e+03, 3.9e+03, 1.7e+03,-2.3e+03,
       -3.4e+03,-3.4e+02,-2.3e+03,-1.2e+02, 4.0e+03,
        3.3e+03,-1.8e+03,-2.6e+02,-1.1e+03, 2.6e+03,
       -2.3e+03, 2.7e+03, 2.8e+01, 1.6e+03,-1.4e+03,
                                                      ]
Accuracy.
    tr_accuracy = 98.2
    te_accuracy = 92.5

::::::::::::::::::::
 TARGET NUMBER 4
::::::::::::::::::::
w* = [
       -1.9e+02,-1.6e+02, 1.2e+02,-5.1e+01,-4.8e+02,
        7.4e+01, 2.0e+02,-1.0e+02, 2.1e+02,-3.1e+02,
        3.1e+02, 4.4e+01,-2.5e+02, 2.5e+02,-2.5e+02,
        2.5e+02,-1.5e+02,-7.7e+01, 7.5e+00,-1.1e+02,
        2.7e+02, 2.9e+02, 5.0e+00, 9.6e+01, 4.7e+02,
       -9.4e+01,-3.4e+02,-1.0e+02, 2.7e+02,-1.8e+02,
       -2.7e+02,-1.4e+02,-2.3e+02, 2.5e+02,-9.3e+01,
                                                      ]
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 99.2

::::::::::::::::::::
 TARGET NUMBER 5
::::::::::::::::::::
w* = [
        1.2e+02,-5.8e+01, 1.7e+02, 2.0e+02, 2.7e+02,
        7.4e+01,-1.7e+02,-1.6e+02,-4.0e+01,-3.3e+02,
        4.1e+02, 4.9e+02, 3.7e+02, 3.2e+02,-3.0e+02,
       -3.5e+01,-2.0e+02,-3.8e+01,-1.6e+02, 3.1e+02,
       -3.3e+02,-1.5e+02,-1.4e+02, 1.9e+01,-5.5e+01,
       -5.0e+01,-7.6e+01,-1.2e+02,-5.6e+01, 1.8e+01,
       -8.5e+01,-1.5e+02,-1.8e+02, 4.0e+01,-2.2e+02,
                                                      ]
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 97.5

::::::::::::::::::::
 TARGET NUMBER 6
::::::::::::::::::::
w* = [
       -9.5e+02,-1.9e+03, 1.6e+03,-3.7e+02, 2.9e+03,
       -2.1e+03, 1.6e+03,-1.1e+03,-1.5e+03,-2.1e+03,
        1.1e+03,-1.3e+03,-1.1e+03,-2.5e+03,-3.6e+03,
        2.8e+03, 2.4e+03, 1.5e+03, 1.4e+03,-1.2e+03,
        3.0e+03,-2.3e+03,-2.0e+03,-2.3e+03, 1.2e+03,
        1.7e+03,-6.7e+02,-9.1e+02,-2.3e+03, 1.0e+03,
       -2.4e+02, 8.2e+02, 1.6e+03, 1.2e+03, 8.4e+01,
                                                      ]
Accuracy.
    tr_accuracy = 100.0
    te_accuracy = 98.6

::::::::::::::::::::
 TARGET NUMBER 7
::::::::::::::::::::
w* = [
        1.9e+03, 1.1e+03, 1.1e+03, 1.9e+03, 8.6e+01,
       -1.7e+03,-8.7e+02,-3.7e+02, 4.0e+01, 1.2e+03,
        2.0e+01,-4.5e+01,-6.5e+02, 5.5e+01, 9.8e+02,
       -2.1e+02,-2.5e+02,-6.0e+02, 5.5e+02, 8.4e+01,
       -6.7e+02,-3.8e+02, 9.5e+02,-3.4e+02,-5.7e+02,
       -8.1e+02, 8.3e+02, 4.9e+02,-2.8e+02, 2.3e+02,
        1.2e+03,-1.0e+03,-1.9e+03,-1.9e+03,-1.9e+03,
                                                      ]
Accuracy.
    tr_accuracy = 99.8
    te_accuracy = 98.3

::::::::::::::::::::
```

```
 TARGET NUMBER 8
::::::::::::::::::::
w* = [
       -2.4e+03, 1.7e+03, 7.4e+03, 3.2e+03,-1.0e+04,
        3.4e+03,-1.2e+04,-9.2e+03,-3.7e+03, 7.3e+03,
        6.2e+03,-1.5e+04,-7.3e+03,-1.0e+04, 8.0e+03,
       -1.2e+04, 1.6e+04, 1.2e+04, 8.7e+03,-1.2e+04,
        2.1e+04,-4.8e+03,-3.2e+02,-9.0e+03, 8.6e+03,
        1.1e+04,-1.1e+04,-7.4e+03,-6.9e+03, 4.8e+03,
       -8.6e+03,-1.4e+03, 4.4e+03, 7.2e+03,-1.5e+04,
                                                    ]
Accuracy.
    tr_accuracy = 97.2
    te_accuracy = 90.2

::::::::::::::::::::
 TARGET NUMBER 9
::::::::::::::::::::
w* = [
       -1.7e+03, 6.2e+03, 2.7e+03, 6.3e+02,-7.8e+03,
        7.4e+02,-4.6e+03,-5.5e+03,-4.2e+02, 2.5e+03,
        7.2e+03,-5.8e+03,-5.1e+02,-3.3e+03,-1.6e+03,
       -2.6e+03, 9.8e+03, 4.9e+03, 6.6e+03, 9.9e+03,
       -9.1e+03,-2.8e+03,-1.0e+03,-2.0e+03, 7.5e+03,
       -5.0e+03,-8.3e+03,-2.3e+02,-7.5e+03, 2.3e+03,
       -1.7e+03, 5.0e+03, 5.1e+02,-2.4e+02,-7.0e+03,
                                                    ]
Accuracy.
    tr_accuracy = 98.8
    te_accuracy = 91.7

::::::::::::::::::::
 TARGET NUMBER 0
::::::::::::::::::::
w* = [
       -4.1e+02, 3.1e+02, 4.2e+02,-2.6e+02,-4.8e+02,
        3.3e+02,-2.4e+02,-5.7e+02,-1.4e+03, 7.0e+02,
        6.8e+02,-1.3e+03,-2.7e+02,-7.3e+02, 1.1e+03,
        1.9e+03,-2.0e+03,-1.2e+03,-1.9e+03, 1.0e+03,
        1.7e+03, 3.4e+01,-1.8e+02,-2.1e+02, 5.5e+02,
        8.2e+02,-1.9e+02, 1.4e+01,-1.0e+03, 3.3e+02,
       -6.2e+02, 7.4e+02, 5.2e+02, 6.9e+02,-8.2e+02,
                                                    ]
Accuracy.
    tr_accuracy = 99.6
    te_accuracy = 95.9
```



This last plot is a representation of the optimal weights for all the numbers, so the darker the "pixel" is, means that it will be less important is at the end to identify the number (and have a lower weight), the brigther the "pixel" is, more important it is at the decision time (and have a higher weight).

To try and explain the diferences we can show all the Accuracy measures in a table:

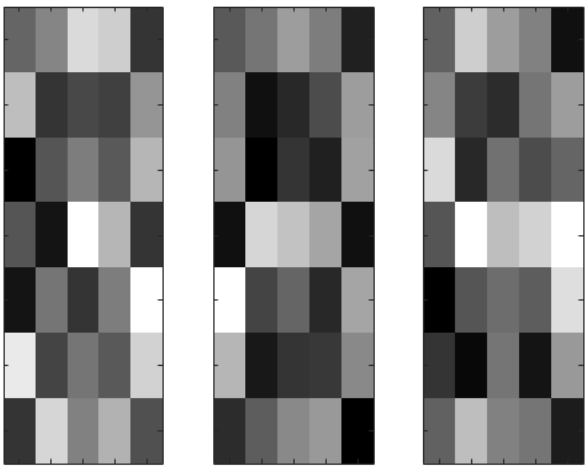
```
target_number = {'1';'2';'3';'4';'5';'6';'7';'8';'9';'0'};
TrainingAccuracy = TrainingAccuracy';
TestAccuracy = TestAccuracy';

T = table(TrainingAccuracy,TestAccuracy,'RowNames',target_number)
```

```
T =

  10×2 table

         TrainingAccuracy    TestAccuracy
         ________________    ____________

    1           100              99.78
```

```
2          100            97.62
3         98.2            92.46
4          100             99.2
5          100            97.48
6          100            98.62
7         99.8             98.3
8         97.2             90.2
9         98.8            91.74
0         99.6            95.92
```

We can see that the numbers with the lowest Test Accuracy are 3, 8 and 9. To try and see the reason, we can plot the optimal weigths in order to see similarities within them.

```
subplot(1,3,1);
numplot(Wk(:,2),1);
subplot(1,3,2);
numplot(Wk(:,7),1);
subplot(1,3,3);
numplot(Wk(:,8),1);
```



It is clear than all 3 optimal weigths are very similar, that is because all three numbers have similar sections, so when the method is traing to identify either an 8 or a 9, it is going to struggle to get it right.

Published with MATLAB® R2018b