Unconstrained optimization

GECD / MIRI - BarcelonaTech

Lab Assignment
Pattern recognition with Single Layer Neural Network (SLNN)

v3.1-2003

F.-Javier Heredia

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

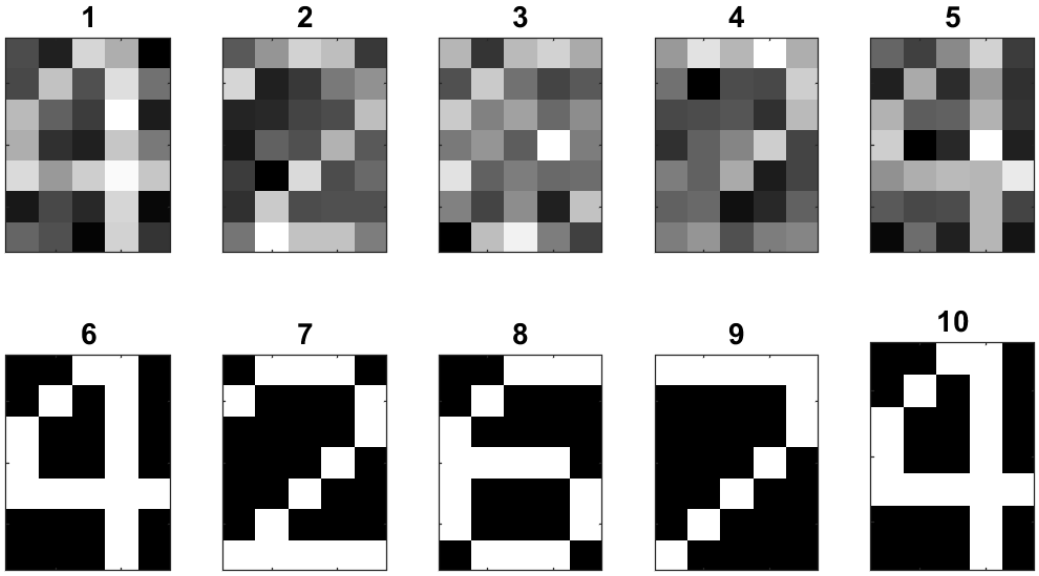
Departament d'Estadística
i Investigació Operativa

Pattern recognition with SLNN

1. Presentation.
2. Single Layer Neural Network (SLNN).
   - Architecture and loss function.
   - Training and testing.
   - Gradient of the loss function.
   - Backtracking line search.
3. Pattern recognition with SLNN.
   - Problem statement and modelling.
   - Generation of training and test data sets.
   - Coding the loss function and gradient.
4. Assignment.
   - Part 1: pattern recognition with GM, CGM and QNM.
   - Part 2: pattern recognition with stochastic gradient.
   - Part 3: computational study of the performance.
   - Report.
5. Summary of supporting codes.

Presentation

- The aim of this project is to develop an application, based on the unconstrained optimization algorithms studied in this course, that allow to recognize the numbers in a sequence of blurred digits:

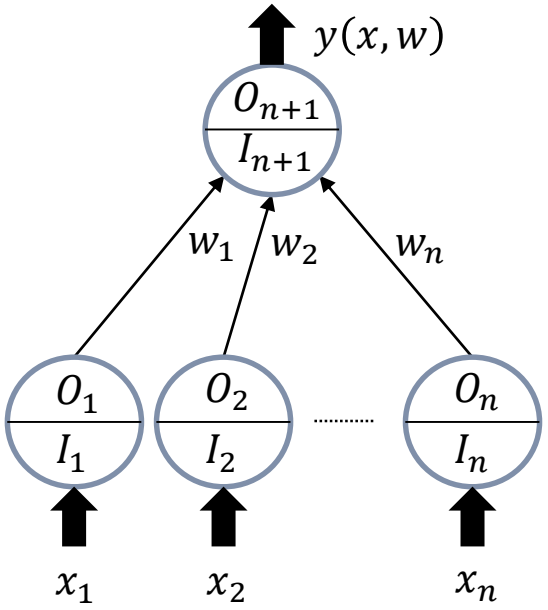


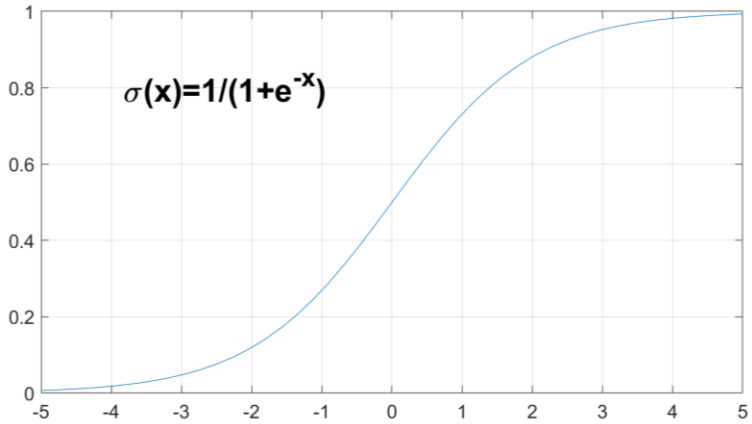
Original blurred numbers

Identified by the neural network

- The procedure to achieve that goal will be to formulate a **Single Layer Neural Network** that is going to be **trained to recognize** the different numbers with **First Derivative Optimization methods**.

Single layer Neural Network (SLNN): architecture





- **Input signal:**

$$I_i = x_i, i = 1,2,\dots,n \;\; ; \;\; I_{n+1} = \sum_{i=1}^{n} w_i \cdot O_i$$

- **Activation function (sigmoid function) :**

$$O_i = \sigma(I_i) \;, \qquad \sigma(x) = 1/(1+e^{-x})$$

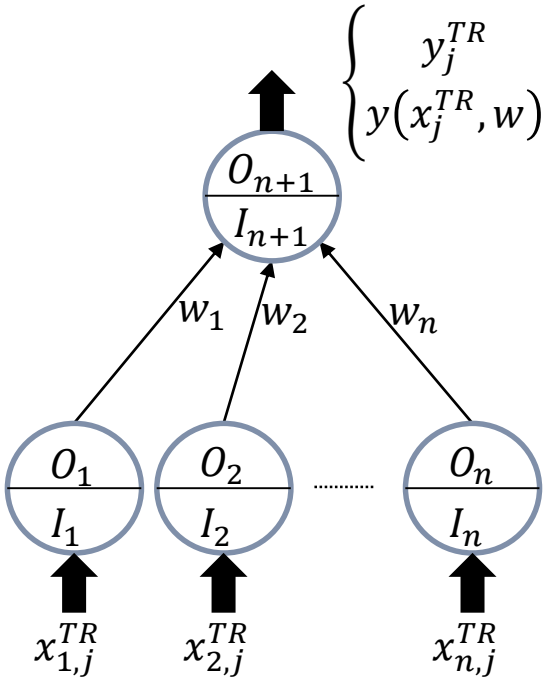
- **Output signal: assumed to be binary**

$$y(x,w) = \quad \sigma(I_{n+1}) = \sigma\left(\sum_{i=1}^{n} w_i O_i\right) = \sigma\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)$$

$$= \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)}\right)^{-1}$$

$$= \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot (1+e^{-x_i})^{-1}\right)}\right)^{-1}$$

SLNN: training



- **Training data set, size p:**

$$X^{TR} = \left[x_1^{TR}, x_2^{TR}, \dots, x_p^{TR}\right] = \begin{bmatrix} x_{1,1}^{TR} & x_{1,2}^{TR} & \cdots & x_{1,p}^{TR} \\ x_{2,1}^{TR} & x_{2,2}^{TR} & \cdots & x_{2,p}^{TR} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1}^{TR} & x_{n,2}^{TR} & \cdots & x_{n,p}^{TR} \end{bmatrix}$$

$$y^{TR} = \begin{bmatrix} y_1^{TR} & y_2^{TR} & \cdots & y_p^{TR} \end{bmatrix}^T$$

- **Loss function**: for a given (X^{TR}, Y^{TR})

$$L(X^{TR}, y^{TR}) = \min_{w \in \mathbb{R}^n} L(w; X^{TR}, y^{TR}) = \sum_{j=1}^{p} \left(y(x_j^{TR}, w) - y_j^{TR}\right)^2$$

- **Loss function with L2 regularization with param. λ:**

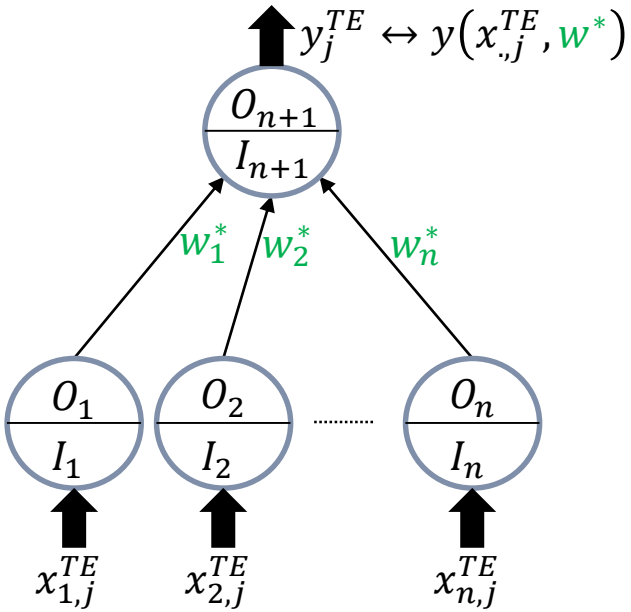
$$\tilde{L}(X^{TR}, y^{TR}, \lambda) = \min_{w \in \mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda) = L(w; X^{TR}, y^{TR}) + \lambda \cdot \frac{\|w\|^2}{2}$$

- **Training accuracy (%)**: $w^* = \text{argmin}_{w \in \mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda)$

$$\text{Accuracy}^{TR} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\underbrace{\left[y\left(x_j^{TR}, w^*\right)\right]}_{\texttt{round()}}, y_j^{TR}}$$

where $\delta_{x,y} = \begin{cases} 1 & if\ x = y \\ 0 & if\ x \neq y \end{cases}$ (*Kronecker delta*).

SLNN : testing



- **Test data set, size q:**

$$X^{TE} = \left[x_1^{TE}, x_2^{TE}, \ldots, x_q^{TE}\right] = \begin{bmatrix} x_{1,1}^{TE} & x_{1,2}^{TE} & \cdots & x_{1,q}^{TE} \\ x_{2,1}^{TE} & x_{2,2}^{TE} & \cdots & x_{2,q}^{TE} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1}^{TE} & x_{n,2}^{TE} & \cdots & x_{n,q}^{TE} \end{bmatrix}$$

$$y^{TE} = \begin{bmatrix} y_1^{TE} & y_2^{TE} & \cdots & y_q^{TE} \end{bmatrix}^T$$

- **Test accuracy (%):**

$$\text{Accuracy}^{TE} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\left[y\left(x_j^{TE}, w^*\right)\right], y_j^{TE}}$$

- **Overfitting**: if $\text{Accuracy}^{TR} \gg \text{Accuracy}^{TE}$

SLNN : gradient (1/2)

- **Loss function (objective function):**

$$\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = \sum_{j=1}^{p} \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right)^2 + \frac{\lambda}{2} \cdot \sum_{i=1}^{n} w_i^2$$

- **Gradient:**

$$\frac{\partial \tilde{L}(w; X^{TR}, y^{TR}, \lambda)}{\partial w_i} = \sum_{j=1}^{p} 2 \cdot \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right) \cdot \frac{\partial y\left(x_j^{TR}, w\right)}{\partial w_i} + \lambda \cdot w_i \quad (1)$$

with

$$y\left(x_j^{TR}, w\right) = \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \left(1 + e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-1} \quad (2)$$

SLNN : gradient (2/2)

- Let us find $\partial y\left(x_j^{TR}, w\right)/\partial w_i$:

$$\frac{\partial y\left(x_j^{TR}, w\right)}{\partial w_i} = \frac{\partial}{\partial w_i}\left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-1} =$$

$$= \overbrace{-\left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-2}}^{-y\left(x_j^{TR},w\right)^2} \cdot \overbrace{e^{-\left(\sum_{i=1}^{n} w_i \cdot \left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}}^{\left(y\left(x_j^{TR},w\right)^{-1}-1\right)} \cdot \left(-\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)$$

$$= y\left(x_j^{TR}, w\right)^2 \cdot \left(y\left(x_j^{TR}, w\right)^{-1} - 1\right) \cdot \left(1 + e^{-x_{i,j}^{TR}}\right)^{-1}$$

$$= y\left(x_j^{TR}, w\right) \cdot \left(1 - y\left(x_j^{TR}, w\right)\right) \cdot \left(1 + e^{-x_{i,j}^{TR}}\right)^{-1}$$

Therefore:

$$\frac{\partial \tilde{L}\left(w; X^{TR}, y^{TR}, \lambda\right)}{\partial w_i} = \sum_{j=1}^{p} 2 \cdot \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right) \cdot y\left(x_j^{TR}, w\right) \cdot \left(1 - y\left(x_j^{TR}, w\right)\right) \cdot \left(1 + e^{-x_{i,j}^{TR}}\right)^{-1} + 2\lambda \cdot w_i$$

SLNN: backtracking linesearch

- The backtracking linesearch algorithm `Alg.BLS` cannot handle conveniently the SLNN problem. We need to introduce two modifications in the computation of the linesearch:
  1. The maximum step length cannot be a constant for every iteration. Instead, it must be updated dynamically using information of the local behaviour of f near the iterated point at each iteration, using some of the formulas (N&W page 58):

$$\alpha_1^{max} = \alpha^{k-1}\frac{\nabla f^{k-1^T} d^{k-1}}{\nabla f^{k^T} d^k};\quad \alpha_2^{max} = \frac{2\left(f^k - f^{k-1}\right)}{\nabla f^{k^T} d^k}.$$

  2. A BLS based on interpolations must be used (see N&W 3.4), as the one proposed in Alg 3.2 and 3.3 of N&W, implemented in function **`uo_BLSNW32`**:

```
function [alpha,iout] =
        uo_BLSNW32(f,g,x,d,almax,c1,c2,kBLSmax,epsal)
```

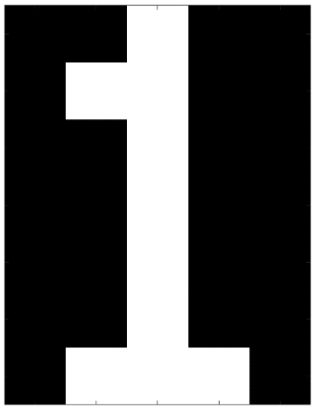
     where **`f,g,d,x,almax,c1,c2`** are as usual, **`iout=0`** if the procedure succeeds and:

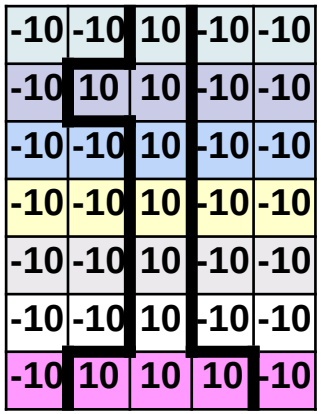
     - **`kBLSmax`** is the maximum number of iterations of the BLS algorithm: if exceeded, the algorithm stops with **`iout=1.`**
     - **`epsal`** is the minimum variation between two consecutive reductions of α^k, meaning that the algorithm will stop with **`iout=2`** whenever $\left|\alpha^{k+1} - \alpha^k\right| <$ **`epsal`**.

Pattern recognition with SLNN (1/2)

- We are going to use the SLNN to solve a problem of pattern recognition over a small 7x5 pixels matrix picturing the 10 digits:

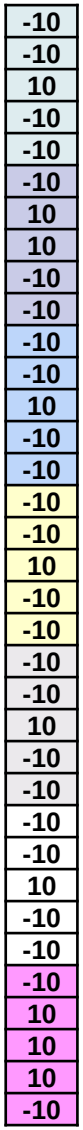
- To obtain the input data of the SLNN x, each white pixel is assigned with a value of 10 and each black pixel with a value of -10 then vectorized and blurred with a Gaussian noise with $\mu = 0$ and $\sigma = \sigma_{rel} \cdot 10$.





Vectorization

$x =$

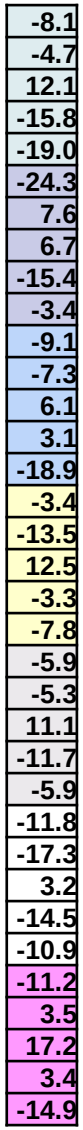


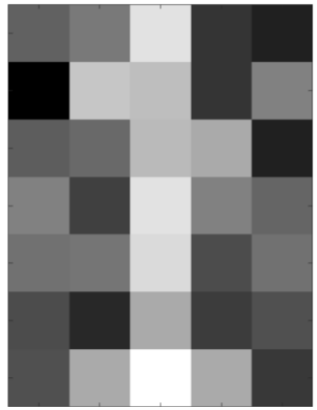
Gaussian blur

$x \leftarrow x + \epsilon =$

$\epsilon \sim N(0, 5)$

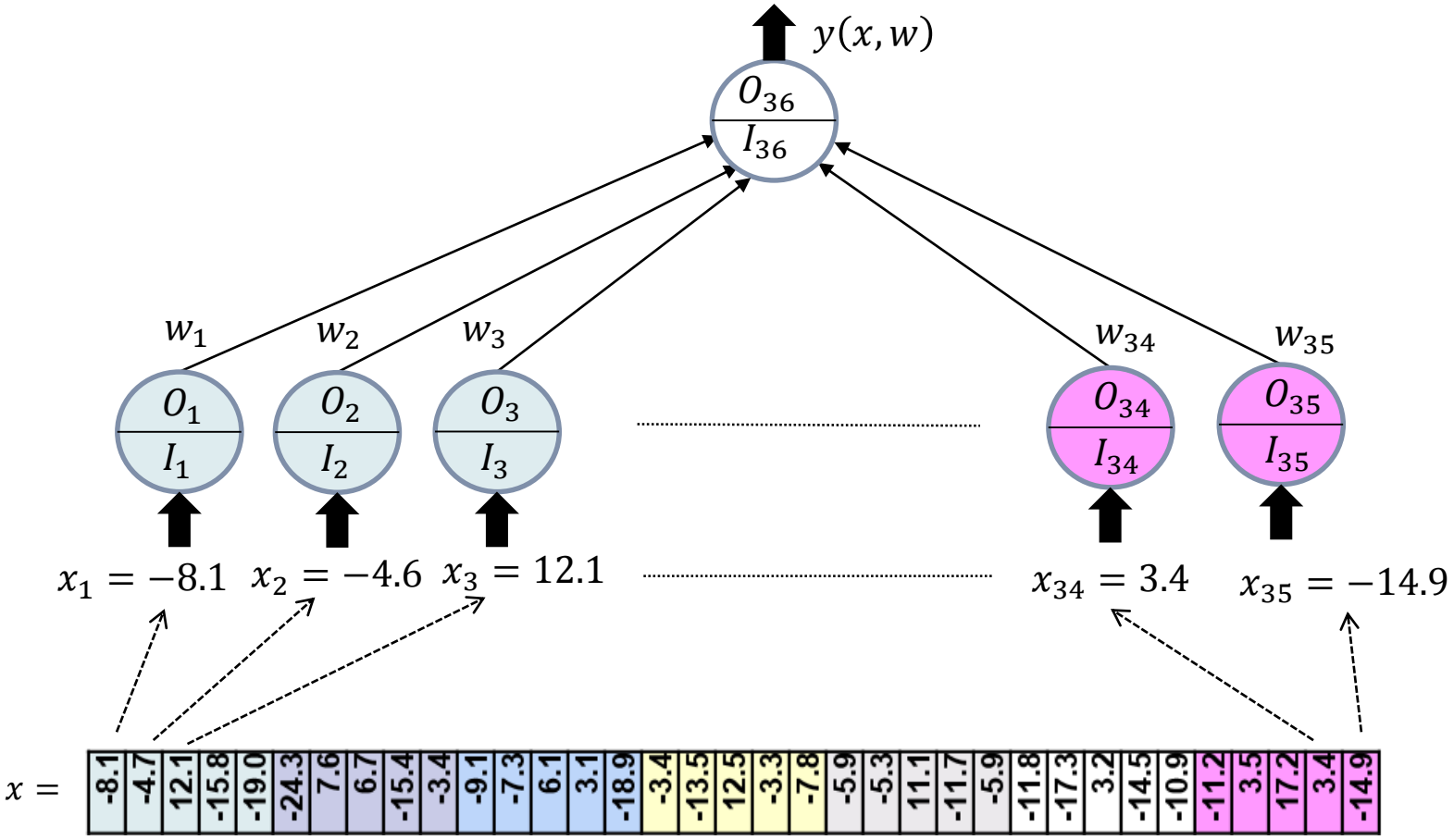
$\sigma_{rel} = 0.5$





Pattern recognition with SLNN (2/2)

- The resulting vectorised and blurred digit x are going to be taken as the inputs of a SLNN:



Training and test data set (1/2)

- The objective of the SLNN is to recognize a set of target numbers, **num_target**, for instance **num_target = [1 3 5 7 9]** will recognize the odd numbers between 0 and 9.
- To this end, the training and test data sets:

$$X^{TR} = \left[x_1^{TR}, x_2^{TR}, \ldots, x_p^{TR}\right] \equiv \texttt{Xtr(1:35,1:tr\_p)} \text{ and } y^{TR} \equiv \texttt{ytr(1:tr\_p)}$$

$$X^{TE} = \left[x_1^{TE}, x_2^{TE}, \ldots, x_p^{TE}\right] \equiv \texttt{Xte(1:35,1:te\_q)} \text{ and } y^{TE} \equiv \texttt{yte(1:te\_q)}$$

must be generated with the help of function

```
function [X,y] = uo_nn_dataset(seed, size, target, freq)
```

This function will generate a dataset, where:

- **X,y** are the generated data sets (**Xtr, ytr** or **Xte, yte**).
- **seed** is the seed for the Matlab random numbers generator. The numbers in the dataset are randomly choosed, guaranteeing a frequency of the digits in **target** close to **freq**. The value σ_{rel} for each digit is also randomly selected within the range $[0.25, 1]$.
- **size** is the size of the data set (number of columns/elements of array **X/y**).
- **target** is the set of digits to be identified.
- **freq** is the frequency of the digits **target** in the data ser. For instance, if **target=[1 2]** and **freq=0.5**, the digits 1 and 2 will be, approximately, half of the total digits in the data set **X**. If **freq≤0.0**, every digit is (approx.) equally represented in the dataset.

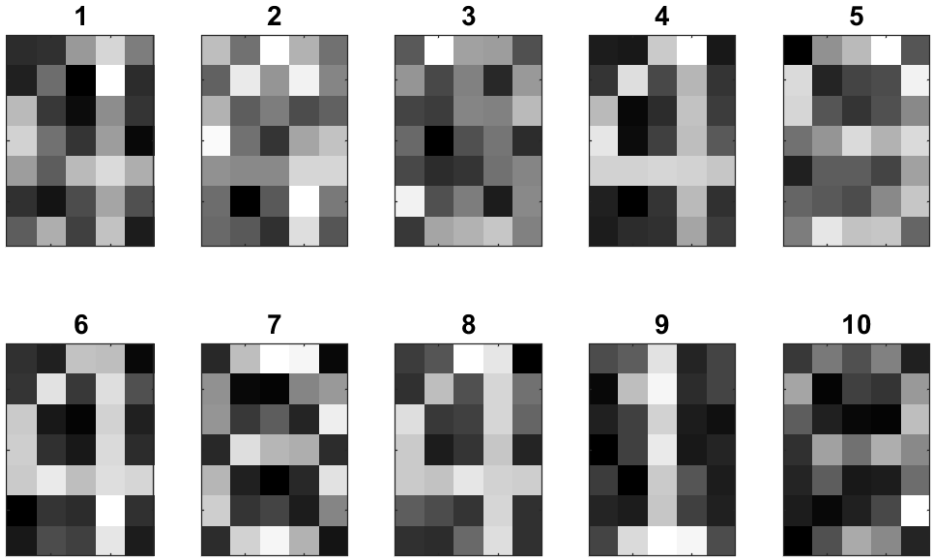
Training and test data set (2/2)

- For instance, let **`seed=1234, size=10, target=[4], freq=0.5`** then:

```
>> [X,y]=uo_nn_dataset(1234,10,[4],0.5)
X =
  -11.5323    8.5784   -8.2363  -11.4127  -31.3497   -9.9915  -10.0134   -9.8603   -5.8843   -5.6767
  -11.0925   -6.6966   46.1249  -11.9861    2.8732  -12.0420    8.8078   -6.9858   -4.5738    6.4063
    3.5013   21.9754   15.0901    9.6655   11.6102    7.1156   17.6479   14.5913    9.2044   -1.3036
.....................................................................................................
 -13.9129  -12.5358    5.2724   -7.4084   -7.6072  -12.5149  -12.5704  -11.6329   -6.2185   -9.5339
y =
     1     1     0     1     0     1     0     1     0     0
```

and the graphical representation:

```
>> uo_nn_Xyplot(X,y,[])
```



- **`function uo_nn_Xyplot(X,y,w)`**

  plots a set of vectorised digits, and the recognition brought by a vector *w*:

  - **`X`** an array of vectorised digits.
  - **`y`** associated output of the SLNN.
  - **`w`** vector of weights *w* (optional).

Loss function and its gradient (1/2)

- Let **`Xtr,ytr`** be the training data set:

```
[Xtr,ytr] = uo_nn_dataset(tr_seed, tr_p, num_target, tr_freq, noise_freq);
```

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

$$y(X^{TR},w) \stackrel{\text{def}}{=} \left[y(x_1^{TR},w),\ldots,y\left(x_p^{TR},w\right)\right];\ \sigma(X^{TR}) = \begin{bmatrix} \sigma(x_{11}^{TR}) & \cdots & \sigma\left(x_{1p}^{TR}\right) \\ \vdots & \ddots & \vdots \\ \sigma(x_{n1}^{TR}) & \cdots & \sigma\left(x_{np}^{TR}\right) \end{bmatrix}$$

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

$$\tilde{L}(w;X^{TR},y^{TR},\lambda) = \|y(X^{TR},w) - y^{TR}\|^2 + \lambda\frac{\|w\|^2}{2}$$

$$\nabla\tilde{L}(w;X^{TR},y^{TR},\lambda) = 2\sigma(X^{TR})\left((y(X^{TR},w) - y^{TR}) \circ y(X^{TR},w) \circ \left(1 - y(X^{TR},w)\right)\right)^T + \lambda w$$

where ∘ stands for the **element-wise (o *Hadamard*) product**. These expressions can be easily coded in Matlab, taking profit of the **element-wise operators " . / " and " . * "**:

$\sigma(X)$	`sig = @(X) 1./(1+exp(-X));`
$y(X,w)$	`y = @(X,w) sig(w'*sig(X));`
$\tilde{L}$	`L = @(w) norm(y(Xtr,w)-ytr)^2 + (la*norm(w)^2)/2;`
$\nabla\tilde{L}$	`gL = @(w) 2*sig(Xtr)*((y(Xtr,w)-ytr).*y(Xtr,w).*(1-y(Xtr,w)))'+la*w;`

Assignment.

There are **three** different parts in this assignment:

- **Part 1:** to develop a code `uo_nn_solve` for the **recognition** of some specific target with GM, CGM and QNM).
- **Part 2:** to extend code `uo_nn_solve` with the **stochastic gradient (SGM)**.
- **Part 3:** to conduct a **comparation of the performance** of the abovementioned methods (only GM, QNM and SGM).

Part 1: patern recognition

- Matlab script to solve the recognition of a given **num_target**

uo_nn_main.m : recognition of num_target digits.

```
clear;
%
% Parameters for dataset generation
%
num_target =[3];
tr_freq    = .5;
tr_p       = 250;
te_q       = 250;
tr_seed    = 123456;
te_seed    = 789101;
%
% Parameters for optimization
%
la = 1.0;                                                    % L2 regularization.
epsG = 10^-6; kmax = 10000;                                  % Stopping criterium.
ils=3; ialmax = 2; kmaxBLS=30; epsal=10^-3;c1=0.01; c2=0.45; % Linesearch.
isd = 1; icg = 2; irc = 2 ; nu = 1.0;                        % Search direction.
%
% Optimization
%
t1=clock;
[Xtr,ytr,wo,fo,tr_acc,Xte,yte,te_acc,niter,tex]=uo_nn_solve(num_target,
tr_freq,tr_seed,tr_p,te_seed,te_q,la,epsG,kmax,ils,ialmax,kmaxBLS,epsal,c1,c2,isd,icg,irc,nu);
t2=clock;
fprintf(' wall time = %6.1d s.\n', etime(t2,t1));
%
```

Part 1: pattern recognition

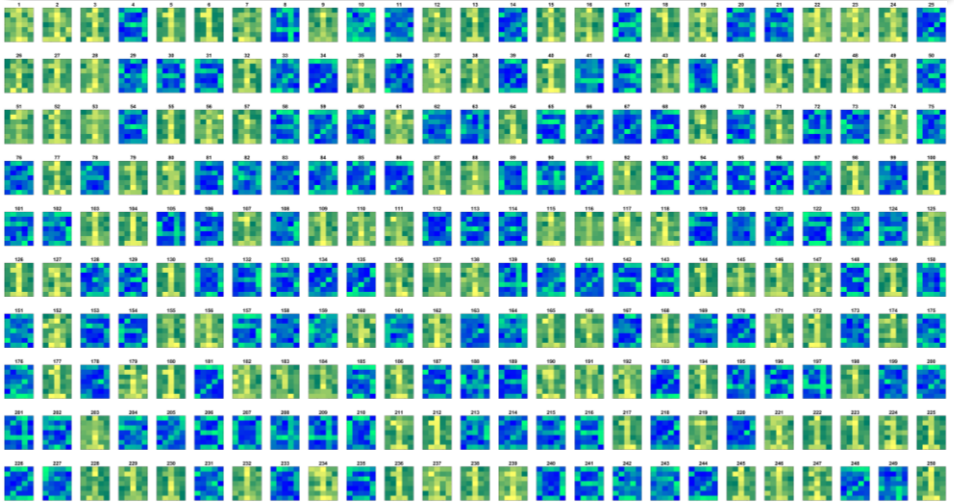
- Function **`uo_nn_solve`** called inside **`uo_nn_main.m`** must solve the instance corresponding to a particular combination of parameters. The actions to be taken inside this function are:
  i. To generate the training data set (X^{TR}, y^{TR}).
  ii. To find the value of w^* minimizing $\tilde{L}(w; X^{TR}, y^{TR}, \lambda)$ with your own optimization routines developed during the course.
  iii. To calculate Accuracy^{TR}.
  iv. To generate the test dataset (X^{TE}, y^{TE}) and to calculate Accuracy^{TE}.

Part 1: example 1, `num_target=[1]`

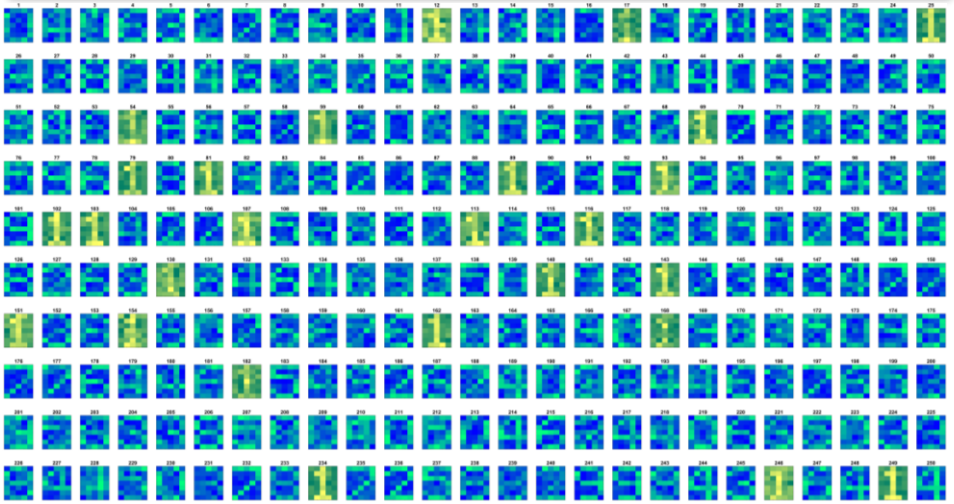
```
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Pattern recognition with neural networks.
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Training data set generation.
[uo_nn]    num_target = 1
[uo_nn]    tr_freq    = 0.50
[uo_nn]    tr_p       = 250
[uo_nn]    tr_seed    = 123456
[uo_nn] Optimization
[uo_nn]    L2 reg. lambda = 0.00
[uo_nn]    epsG= 1.0e-06, kmax= 20000
[uo_nn]    ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn]    c1= 0.01, c2= 0.45, isd= 1
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    1   5.00e-01   0  -7.38e+03   6.25e+01   8.59e+01
[uo_nn]    2   8.19e+03   0  -2.90e-07   1.00e+00   5.39e-04
[uo_nn]    3   2.53e+04   0  -6.18e-07   2.16e-04   7.86e-04
[uo_nn]    4                             3.51e-12   2.67e-11
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    wo=[
[uo_nn]         -5.0e+00,-1.1e+01,+2.1e+00,-1.3e+01,-5.2e+00
[uo_nn]         -4.7e+00,+5.1e+00,+1.5e+01,-1.8e+00,-9.0e+00
[uo_nn]         -8.7e+00,-1.8e+00,+1.4e+01,-9.0e+00,-4.4e+00
[uo_nn]         -1.0e+01,-4.9e+00,+1.3e+01,-1.1e+01,-3.8e+00
[uo_nn]         -1.0e+01,-6.7e+00,+8.0e+00,-7.9e+00,-1.5e+01
[uo_nn]         -2.1e+00,-3.6e+00,+1.4e+01,-6.8e+00,-3.4e+00
[uo_nn]         +2.7e-01,+2.2e+00,+2.3e+00,+5.3e+00,-2.4e+00
[uo_nn]       ]
[uo_nn] Test data set generation.
[uo_nn]    te_q     = 250
[uo_nn]    te_seed  = 789101
[uo_nn] tr_accuracy = 100.0
[uo_nn] te_accuracy = 100.0
```

```
>> uo_nn_Xyplot(wo,0,[])
```

```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



```
>> uo_nn_Xyplot(Xte,yte,wo)
```



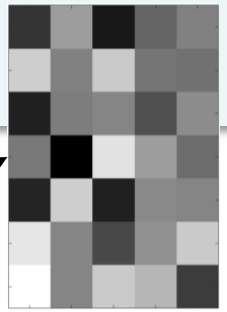
Part 1: example 2, `num_target=[3]`

```
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Pattern recognition with neural networks.
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Training data set generation.
[uo_nn]    num_target = 3
[uo_nn]    tr_freq    = 0.50
[uo_nn]    tr_p       = 250
[uo_nn]    tr_seed    = 123456
[uo_nn] Optimization
[uo_nn]    L2 reg. lambda = 0.00
[uo_nn]    epsG= 1.0e-06, kmax= 10000
[uo_nn]    ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn]    c1= 0.01, c2= 0.45, isd= 1
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    1    1.25e-01   0  -2.45e+03   6.25e+01    4.95e+01
[uo_nn]    2    9.27e-03   2  -1.76e+03   5.43e+01    4.19e+01
[uo_nn]    3    2.39e-02   2  -1.25e+03   3.94e+01    3.54e+01
           .....................................................
[uo_nn] 3731    1.98e+04   0  -1.01e-12   2.94e-06    1.00e-06
[uo_nn] 3732    1.10e+04   0  -1.81e-12   2.93e-06    1.34e-06
[uo_nn] 3733                               2.92e-06    9.96e-07
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    wo=[
[uo_nn]         -1.3e+01,+2.4e+00,-1.7e+01,-5.8e+00,-1.7e+00
[uo_nn]         +9.6e+00,-1.8e+00,+8.7e+00,-3.8e+00,-4.4e+00
[uo_nn]         -1.6e+01,-2.6e+00,-1.5e+00,-9.0e+00,+1.2e-01
[uo_nn]         -3.3e+00,-2.1e+01,+1.2e+01,+2.5e+00,-5.1e+00
[uo_nn]         -1.6e+01,+9.1e+00,-1.6e+01,-7.1e-01,-1.3e+00
[uo_nn]         +1.3e+01,-1.3e+00,-1.0e+01,+7.7e-01,+8.6e+00
[uo_nn]         +1.7e+01,-1.1e+00,+9.0e+00,+5.8e+00,-1.2e+01
[uo_nn]       ]
[uo_nn] Test data set generation.
[uo_nn]    te_q      = 250
[uo_nn]    te_seed   = 789101
[uo_nn] tr_accuracy = 100.0
[uo_nn] te_accuracy = 95.6
```

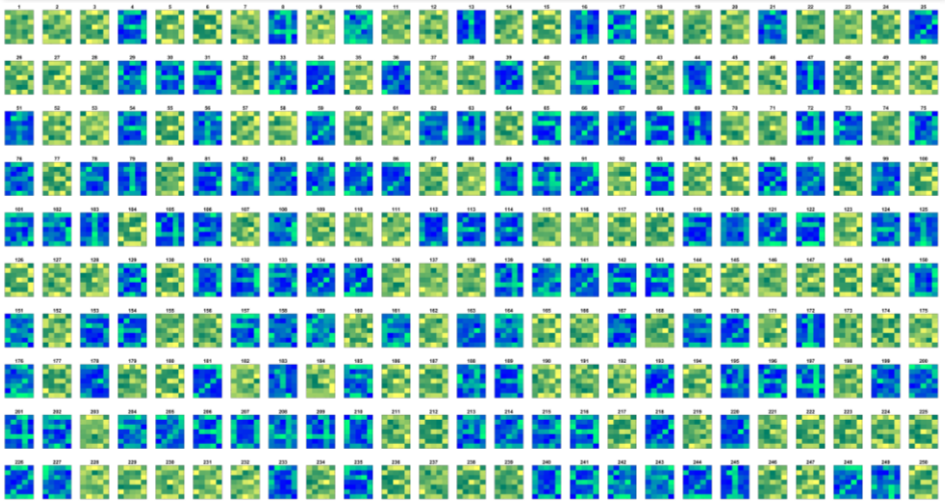
Rigth positive

Rigth negative

```
>> uo_nn_Xyplot(wo,0,[])
```

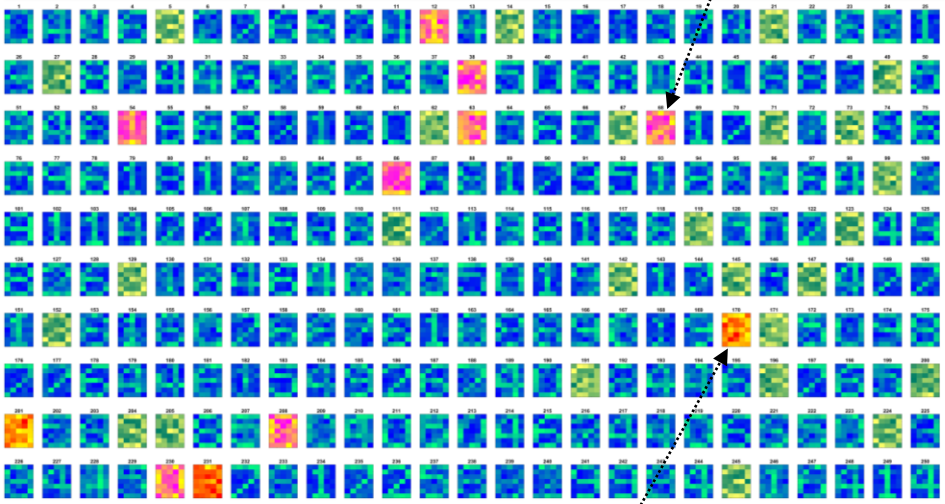


```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



```
>> uo_nn_Xyplot(Xte,yte,wo)
```

False positive

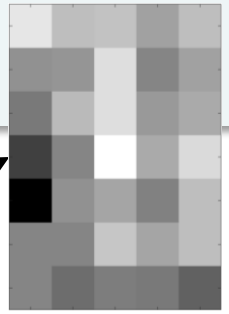


False negative

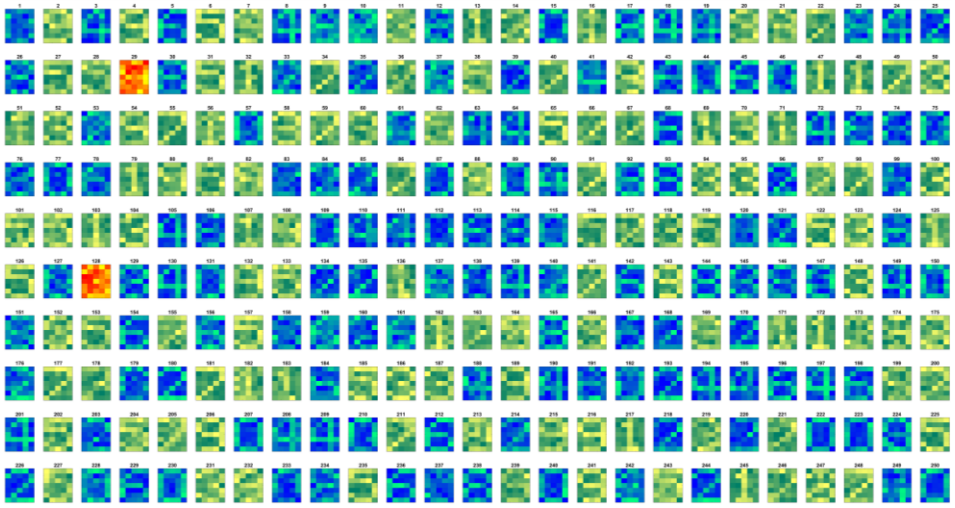
Part 1: example 3, num_target=[1 3 5 7 9]

```
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Pattern recognition with neural networks.
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Training data set generation.
[uo_nn]    num_target = 1 3 5 7 9
[uo_nn]    tr_freq    = 0.50
[uo_nn]    tr_p       = 250
[uo_nn]    tr_seed    = 123456
[uo_nn] Optimization
[uo_nn]    L2 reg. lambda = 1.00
[uo_nn]    epsG= 1.0e-06, kmax= 20000
[uo_nn]    ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn]    c1= 0.01, c2= 0.45, isd= 3
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    1   3.13e-02   0  -1.89e+03   6.25e+01   4.35e+01
[uo_nn]    2   5.25e-03   2  -4.31e+03   5.12e+01   8.16e+01
[uo_nn]    3   4.57e-02   0  -4.10e+02   3.25e+01   5.41e+01
              ..............................................
[uo_nn]   27   1.11e+00   0  -6.80e-12   1.43e+01   4.99e-06
[uo_nn]   28   8.52e-01   0  -1.35e-12   1.43e+01   1.64e-06
[uo_nn]   29                            1.43e+01   4.32e-07
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    wo=[
[uo_nn]         +9.1e-01,+4.3e-01,+4.5e-01,+3.4e-02,+3.9e-01
[uo_nn]         -1.2e-01,-9.8e-02,+8.1e-01,-3.1e-01,+4.1e-02
[uo_nn]         -4.4e-01,+3.6e-01,+7.8e-01,-6.3e-02,+1.8e-01
[uo_nn]         -1.1e+00,-3.0e-01,+1.2e+00,+1.5e-01,+7.5e-01
[uo_nn]         -1.9e+00,-1.4e-01,+9.1e-02,-3.5e-01,+4.0e-01
[uo_nn]         -3.0e-01,-3.0e-01,+5.2e-01,+1.2e-01,+3.8e-01
[uo_nn]         -3.0e-01,-5.7e-01,-3.8e-01,-4.2e-01,-7.4e-01
[uo_nn]       ]
[uo_nn] Test data set generation.
[uo_nn]    te_q      = 250
[uo_nn]    te_seed   = 789101
[uo_nn] tr_accuracy = 99.2
[uo_nn] te_accuracy = 97.2
```

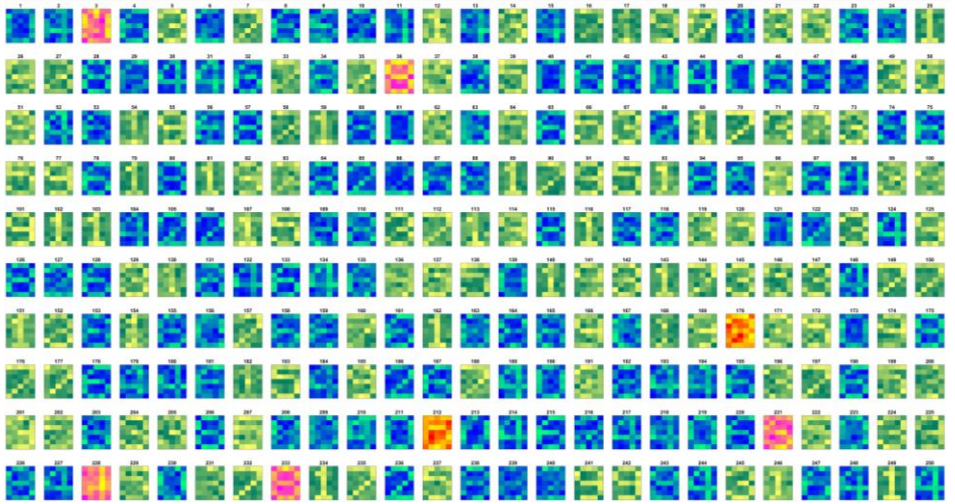
```
>> uo_nn_Xyplot(wo,0,[])
```



```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



```
>> uo_nn_Xyplot(Xte,yte,wo)
```



Part 2: Stochastic Gradient (1/3)

- **Stochastic Gradient Method (SGM).**
  - **Principle:** it is possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d. from the training dataset.
  - **Procedure:**
    i. Sample a minibatch $\boldsymbol{S} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_m}\}$ of $\boldsymbol{m} \ll p$ observations from the training dataset:

$$X_{\boldsymbol{S}}^{TR} = \left[x_{\boldsymbol{s_1}}^{TR}, x_{\boldsymbol{s_2}}^{TR}, \dots, x_{\boldsymbol{s_m}}^{TR}\right];\ y_{\boldsymbol{S}}^{TR} = \left[y_{\boldsymbol{s_1}}^{TR} \quad y_{\boldsymbol{s_2}}^{TR} \quad \cdots \quad y_{\boldsymbol{s_m}}^{TR}\right]^T$$

    ii. Compute search direction through the gradient estimate:

$$d^k \leftarrow -\frac{1}{m} \nabla \tilde{L}(w; X_{\boldsymbol{S}}^{TR}, y_{\boldsymbol{S}}^{TR}, \lambda)$$

    iii. Update the parametres: $w^k \leftarrow w^k + \alpha^k d^k$ with ***learning rate*** $\boldsymbol{\alpha^k}$

$$\alpha^k := \begin{cases} \left(1 - \frac{k}{k^{SG}}\right) \alpha_0^{SG} + \frac{k}{k^{SG}} \alpha^{SG} & \text{if } k \leq k^{SG} \\ \alpha^{SG} & \text{if } k > k^{SG} \end{cases},$$

$$\alpha^{SG} \approx 0{,}01 \cdot \alpha_0^{SG}, k^{SG} := \lfloor \gamma^{SG} \cdot k^{max} \rfloor.$$

Part 2: Stochastic Gradient (2/3)

- Script **`uo_nn_solve.m`** must be extended to include the SGM:

`uo_nn_main.m : recognition of num_target digits.`

```
clear;
%
% Parameters for dataset generation
%
num_target =[3];
tr_freq    = .5;
tr_p       = 250;
te_q       = 250;
tr_seed    = 123456;
te_seed    = 789101;
%
% Parameters for optimization
%
la = 1.0;                                                    % L2 regularization.
epsG = 10^-6; kmax = 10000;                                  % Stopping criterium.
ils=3; ialmax = 2; kmaxBLS=30; epsal=10^-3;c1=0.01; c2=0.45; % Linesearch.
isd = 1; icg = 2; irc = 2 ; nu = 1.0;                        % Search direction.
isg_m = 0.05; isg_al0=2; isg_k=0.3;                          % stochastic gradient
%
% Optimization
%
t1=clock;
[Xtr,ytr,wo,fo,tr_acc,Xte,yte,te_acc,niter,tex]=uo_nn_solve(num_target,
tr_freq,tr_seed,tr_p,te_seed,te_q,la,epsG,kmax,ils,ialmax,kmaxBLS,epsal,c1,c2,isd,isg_m,isg_al0,isg_k,icg,irc,nu);
t2=clock;
fprintf(' wall time = %6.1d s.\n', etime(t2,t1));
%
```

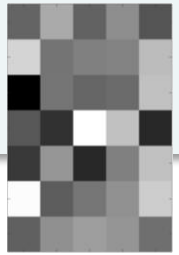
Part 2: Stochastic Gradient (3/3)

```
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Pattern recognition with neural networks (OM/GCED).
[uo_nn] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn] Training data set generation.
[uo_nn]    num_target = 3
[uo_nn]    tr_freq    = 0.50
[uo_nn]    tr_p       = 250
[uo_nn]    tr_seed    = 123456
[uo_nn] Optimization
[uo_nn]    L2 reg. lambda = 1.00
[uo_nn]    epsG= 1.0e-06, kmax= 10000
[uo_nn]    ils= 3, ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn]    c1= 0.01, c2= 0.45, isd= 7
[uo_nn]    sg_m= 0.05, sg_al0= 2.0, sg_ga= 0.30
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    1   2.00e+00   0  -1.46e+02   6.25e+01   4.95e+01
[uo_nn]    2   2.00e+00   0  -7.03e+01   1.63e+02   8.46e+00
[uo_nn]    3   2.00e+00   0  -6.90e+01   1.56e+02   8.31e+00
                .....................................................
[uo_nn] 9998   2.00e-02   0  -4.71e+01   2.68e+01   2.69e+01
[uo_nn] 9999   2.00e-02   0  +1.65e+00   2.64e+01   1.94e+01
[uo_nn]10000                            2.68e+01   2.49e+01
[uo_nn]    k          al  iW       g'*d          f       ||g||
[uo_nn]    wo=[
[uo_nn]         -1.5e-01,+6.0e-02,-1.6e-01,-2.6e-02,-2.0e-01
[uo_nn]         +1.9e-01,-1.0e-01,-6.7e-02,-5.8e-02,+1.1e-01
[uo_nn]         -4.6e-01,-1.0e-01,-1.5e-01,-1.3e-01,+1.3e-01
[uo_nn]         -1.9e-01,-3.1e-01,+3.2e-01,+1.3e-01,-3.4e-01
[uo_nn]         -2.8e-01,-1.3e-03,-3.3e-01,-6.7e-02,+1.3e-01
[uo_nn]         +3.1e-01,-1.7e-01,-9.9e-02,-1.5e-02,+1.7e-01
[uo_nn]         -1.7e-01,-1.7e-02,+2.4e-02,-4.0e-03,-1.2e-01
[uo_nn]       ]
[uo_nn] Test data set generation.
[uo_nn]    te_q     = 250
[uo_nn]    te_seed  = 789101
[uo_nn] tr_accuracy = 96.4
[uo_nn] te_accuracy = 87.2
```

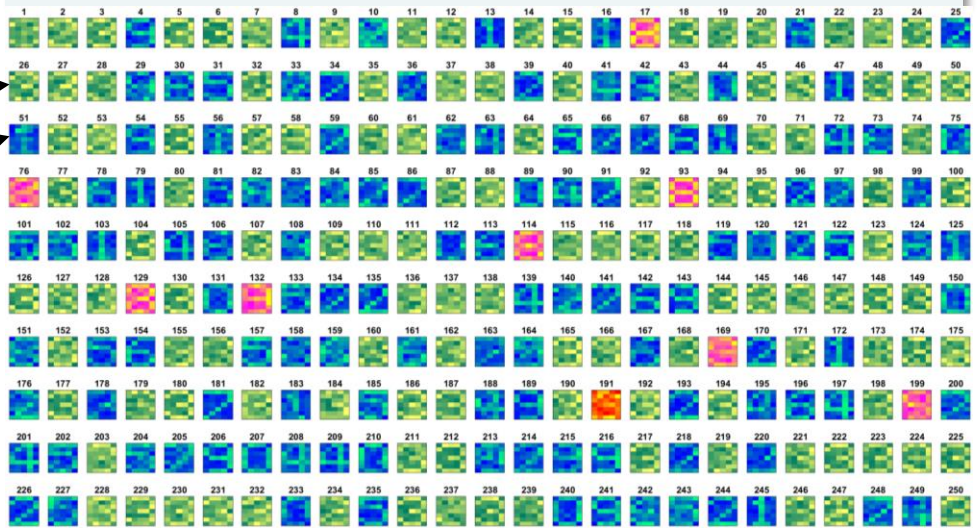
Rigth positive

Rigth negative

```
>> uo_nn_Xyplot(wo,0,[])
```

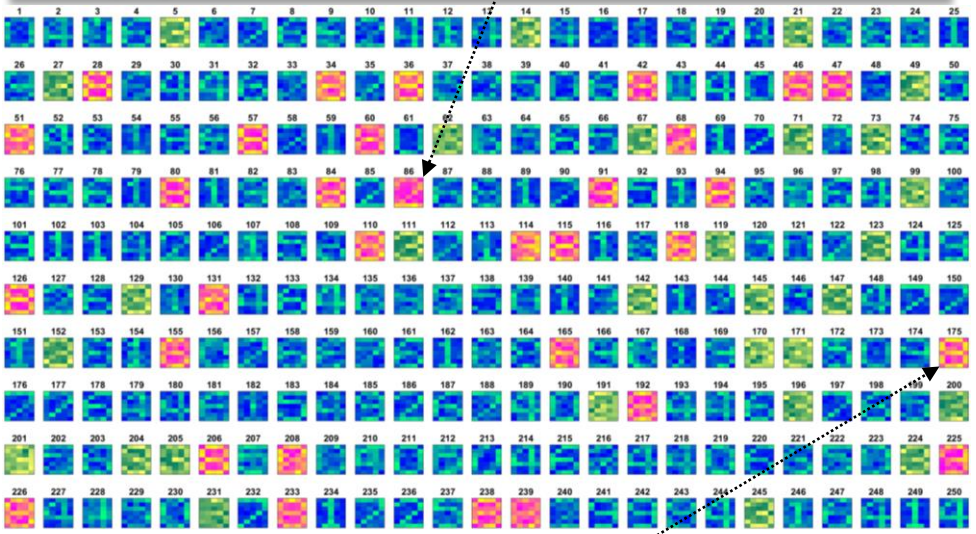


```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



```
>> uo_nn_Xyplot(Xte,yte,wo)
```

False positive



False negative

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Estadística
i Investigació Operativa

F.-Javier Heredia http://gnom.upc.edu/heredia

Part 3: computational study (1/3)

- In this third part we want to conduct a series of computational experiments to study:
  i. How the regularization parameter λ affects the results.
  ii. The relative performance of the different algorithms (GM, QNM,SGM)
- To this end, an instance of the SLNN problem must be solved:
  - For every one of the individual digits, 0 to 9.
  - For every value of the regularization parameter $\lambda \in \{0.0, 1.0, 10.0\}$.
  - For every optimization algorithm: GM, QNM and SGM.

  That makes a total of $10 \times 3 \times 4 = 120$ instances to be solved.

Part 3: computational study (2/3)

- To organize the computational experiments you can use function **`uo_nn_batch.m`**:

`uo_nn_batch.m` : run a batch of SLNN instances.

```
function uo_nn_batch(tr_seed,te_seed)
% Parameters.
tr_p = 250; te_q = 250; tr_freq = .5;                                % Datasets generation
epsG = 10^-6; kmax = 1000;                                           % Stopping criterium.
ils=3; ialmax = 2; kmaxBLS=10; epsal=10^-3; c1=0.01; c2=0.45; % Linesearch.
icg = 2; irc = 2 ; nu = 1.0;                                         % Search direction.
isg_ga1 = 0.05; isg_al0=2; isg_ga2=0.3;                              % stochastic gradient
% Optimization
iheader = 1;
csvfile = strcat('uo_nn_batch_',num2str(tr_seed),'-',num2str(te_seed),'.csv');
fileID = fopen(csvfile ,'w');
t1=clock;
for num_target = 1:10
    for la = [0.0, 1.0, 10.0]
        for isd = [1,3,7]
            [Xtr,ytr,wo,fo,tr_acc,Xte,yte,te_acc,niter,tex]=uo_nn_solve(num_target,
tr_freq,tr_seed,tr_p,te_seed,te_q,la,epsG,kmax,ils,ialmax,kmaxBLS,epsal,c1,c2,isd,isg_ga1,isg_al0,isg_ga2,i
cg,irc,nu,iheader);
            if iheader == 1 fprintf(fileID,'num_target;   la; isd; niter;     tex; tr_acc; te_acc;  L*\n');
end
            fprintf(fileID,'         %1i; %4.1f; %1i;  %4i; %7.4f;  %5.1f;  %5.1f;  %8.2e;\n',
mod(num_target,10), la, isd, niter, tex, tr_acc, te_acc, fo);
            iheader=0;
        end
    end
end
t2=clock;
fprintf(' wall time = %6.1d s.\n', etime(t2,t1));
fclose(fileID);
%
end
```

Part 3: computational study (2/3)

- The outcome of **`uo_nn_batch.m`** is the file **`uo_nn_batch_tr_seed-te_seed.csv`** with the following content:

```
num_target;   la; isd; niter;     tex; tr_acc; te_acc;        L*;
         1;  0.0;   1;     4;  0.0141;  100.0;  100.0;  3.51e-12;
         1;  0.0;   3;    11;  0.0313;  100.0;  100.0;  2.15e-09;
         1;  0.0;   7;  1000;  0.2203;  100.0;  100.0;  2.07e-03;
         1;  1.0;   1;    97;  0.1717;  100.0;  100.0;  3.90e+00;
         1;  1.0;   3;    20;  0.0550;  100.0;  100.0;  3.90e+00;
         1;  1.0;   7;  1000;  0.2640;  100.0;  100.0;  1.16e+01;
.....................................................
         0; 10.0;   1;   110;  0.1478;  100.0;   99.2;  2.54e+01;
         0; 10.0;   3;    43;  0.0907;  100.0;   99.2;  2.54e+01;
         0; 10.0;   7;  1000;  0.2260;   59.2;   26.0;       Inf;
```

The data included in this file will be the base for the analysis of the performance of the different optimization algorithms that you have to perform in this project.

Report (1/3)

- Based on the data gathered in file **`uo_nn_batch.csv`**, perform the following study:

1) **Convergence of the algorithms:** first, we are going to study the global and local convergence of the three algorithms only in terms of the objective function $\tilde{L}$:

a) **Global convergence**: show the value at the optimal solution of the loss function $\tilde{L}$ for every combination λ-algorithm. Are the three algorithms able to identify a stationary solution of the loss function $\tilde{L}$?

b) **Local convergence**: compare the speed of convergence of the three algorithms in terms of the execution time and number of iterations (**`niter`** and **`tex`**). Analyse the running time per iteration (**`tex/niter`**) and try to find an explanation for the different values among the three algorithms.

c) Finally, according to the previous study, discuss the general performance of the three algorithms and justify which is the most efficient for the minimization of $\tilde{L}$.

Report (2/3)

2) **Recognition accuracy**: now, we are going to analyse the recognition accuracy of the SLNN. To this end, you have to perform the following studies providing numerical evidences (averages, plots, tables,…), for both the mean value of $Accuracy^{TE}$ and the individual value for every digit 0 to 9:

a) Analyse the performance of the different combinations λ-algorithm in terms of $Accuracy^{TE}$. Tell if there is some specific combination λ-algorithm that fails to give a proper recognition. According to the numerical experiments, what would be the recommended λ-algorithm combination in terms of, both, the recognition capacity and the speed of convergence?

b) In view of the precedent study, is there any digit specially difficult to identify, regardless of the λ-algorithm combination? For the digit with the overall worst value of $Accuracy^{TE}$, take some representative λ-algorithm instance and display the results with the help of function **`uo_nn_Xyplot`** and try to guess the reasons for the bad recognition accuracy.

Report (3/3)

- This assignment must be done in groups of two. Use a value of **`tr_seed`** and **`te_seed`** based on your student's ID number. You must upload to Atenea a file with the name **`surname-student-1_surname-student-2.zip`** containing:
  - A report (.pdf file) with your answers to the different sections of tasks 1) "Convergence" and 2) "Accuracy".
  - The report must have a cover with the name of the two students and the values of **`tr_seed`** and **`te_seed`.**
  - The source of all the codes used to do the assignment. The codes must be self-contained and must allow the replication of all the results included in the report.

Summary of supporting codes

Function/script	Page
`function [alpha,iout] = uo_BLSNW32(f,g,x,d,almax,c1,c2,kBLSmax,epsal)`: Algorithm 3.2 of Nocedal & Wright (backtracking line search with SWC and curve fitting).	9
`function [X,y] = uo_nn_dataset(seed, ncol, target, freq)`: generates the dataset `X,y`.	12
`function uo_nn_Xyplot(X,y,w)`: plots the dataset `X,y`. If `w` is not `[]`, the plot tells right from wrong predictions of the SLNN.	13

`sig = @(X) 1./(1+exp(-X));`
`y = @(X,w) sig(w'*sig(X));`
`L = @(w) norm(y(Xtr,w)-ytr)^2 + (la*norm(w)^2)/2;`
`gL = @(w) 2*sig(Xtr)*((y(Xtr,w)-ytr).*y(Xtr,w).*(1-y(Xtr,w)))'+la*w;`
	14

`uo_nn_main.m`: run a single recognition.	16/23
`function uo_nn_batch(tr_seed,te_seed)`: run a batch of recognitions.	26