Unconstrained optimization

GECD / MIRI - BarcelonaTech

Lab Assignment

Pattern recognition with Single Layer Neural Network (SLNN)

v4.0-02/21

F.-Javier Heredia

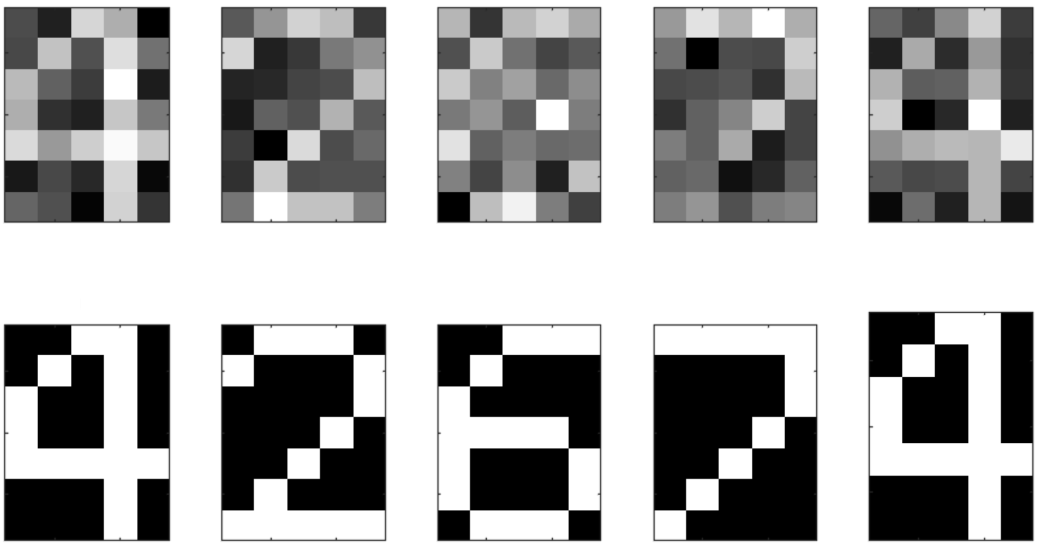
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Estadística
i Investigació Operativa

Pattern recognition with SLNN

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

Sequence=42674 ; Identified=42674

- 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**.

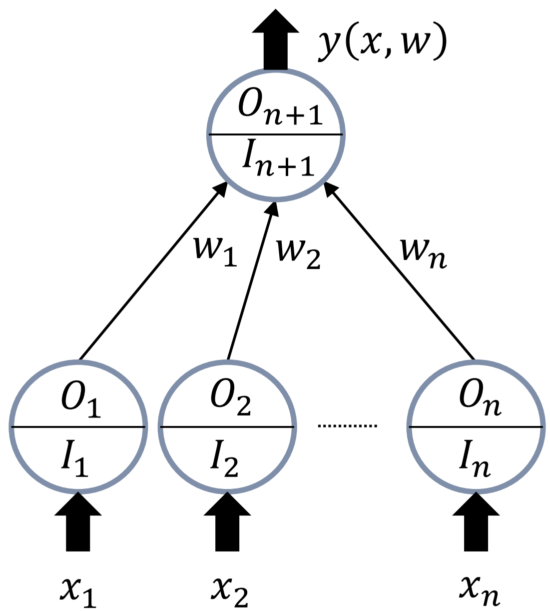
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Estadística i Investigació Operativa

Summary of supporting material

Documents
`uo_nn_v40.pdf`: this document.

Function/script	**Page**
`uo_nn_main.m`: run a single recognition.	link
`uo_nn_batch`: run a batch of recognitions.	link
`function [X,y] = uo_nn_dataset(seed, ncol, target, freq)`: generates the dataset `X,y`.	link
`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.	link
`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).	link

Single layer Neural Network (SLNN): architecture



- **Input signal:**

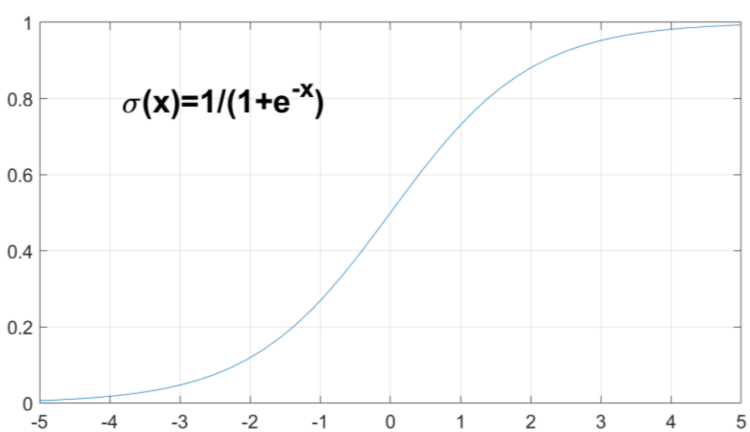
$$I_i = x_i, i = 1,2,\dots,n \quad ; \quad I_{n+1} = \sum_{i=1}^{n} w_i \cdot O_i$$

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

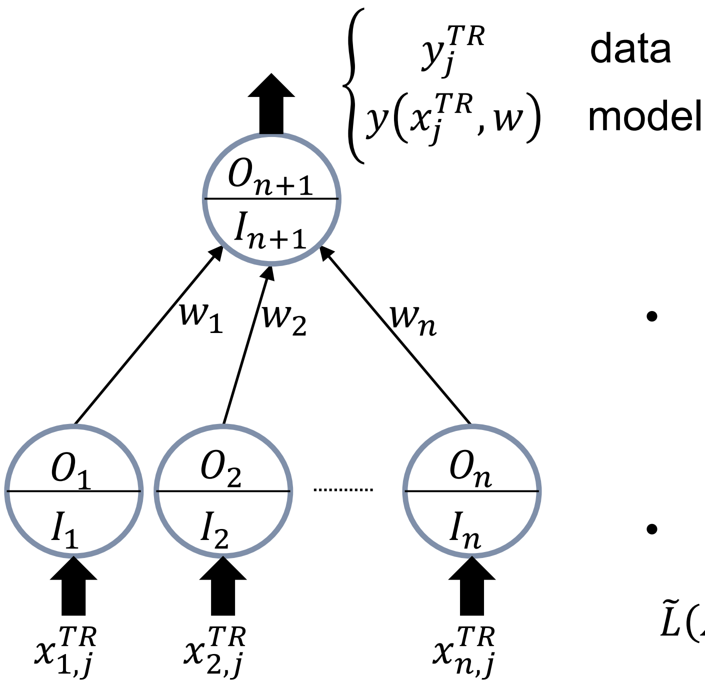
$$O_i = \sigma(I_i) \;, \qquad \sigma(x) = 1/(1+e^{-x})$$

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

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



SLNN: training



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

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

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

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

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

- ***L^2* regularization or *weight decay* (1):**

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

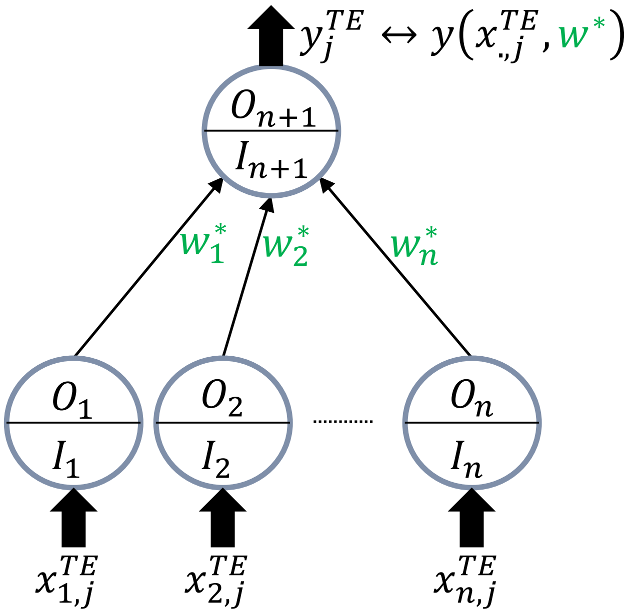
(1): AKA *ridge regression* or *Tikhonov regularization*.

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

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

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

SLNN : testing



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

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

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

- **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)

- **Mean loss function with regularization (objective function):**

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

- **Gradient:**

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

with

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

SLNN : gradient (2/2)

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

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

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

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

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

Therefore:

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

SLNN: backtracking line search

- The backtracking line search algorithm `Alg.BLS` cannot handle conveniently the SLNN problem. We need to introduce two modifications in the computation of the line search:
  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`**:

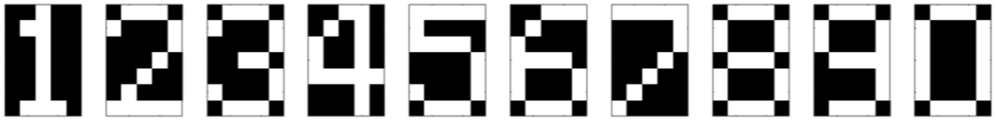
  ```
  function [alpha,iout] =
          uo_BLSNW32(f,g,x,d,almax,c1,c2,kBLSmax,epsal)
  ```

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

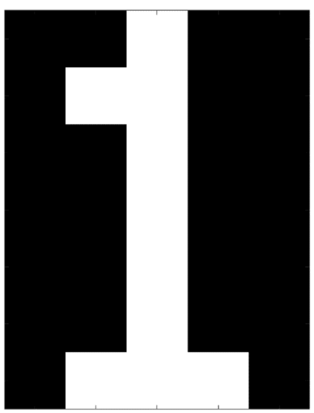
  - **`kBLSmax`** is the maximum number of iterations of the BLS algorithm: if exceeded, the algorithm stops with **`iout=1.`**
  - **`epsal`** is the minimum variation between two consecutive reductions of α^k, meaning that the algorithm will stop with **`iout=2`** whenever $\left|\alpha^{k+1} - \alpha^k\right|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

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 $\sigma = \sigma_{rel} \cdot 10$.



-10	-10	10	-10	-10
-10	10	10	-10	-10
-10	-10	10	-10	-10
-10	-10	10	-10	-10
-10	-10	10	-10	-10
-10	-10	10	-10	-10
-10	10	10	10	-10

Vectorization

$x =$

-10
-10
10
-10
-10
-10
10
10
-10
-10
-10
-10
10
-10
-10
-10
-10
10
-10
-10
-10
-10
10
-10
-10
-10
-10
10
-10
-10
-10
10
10
10
-10

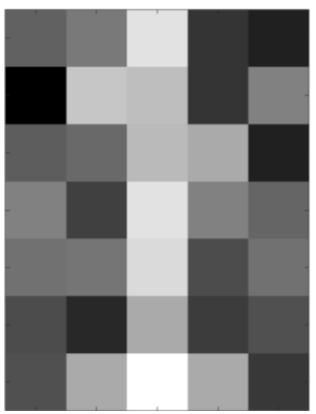
Gaussian blur

$x \leftarrow x + \epsilon =$

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

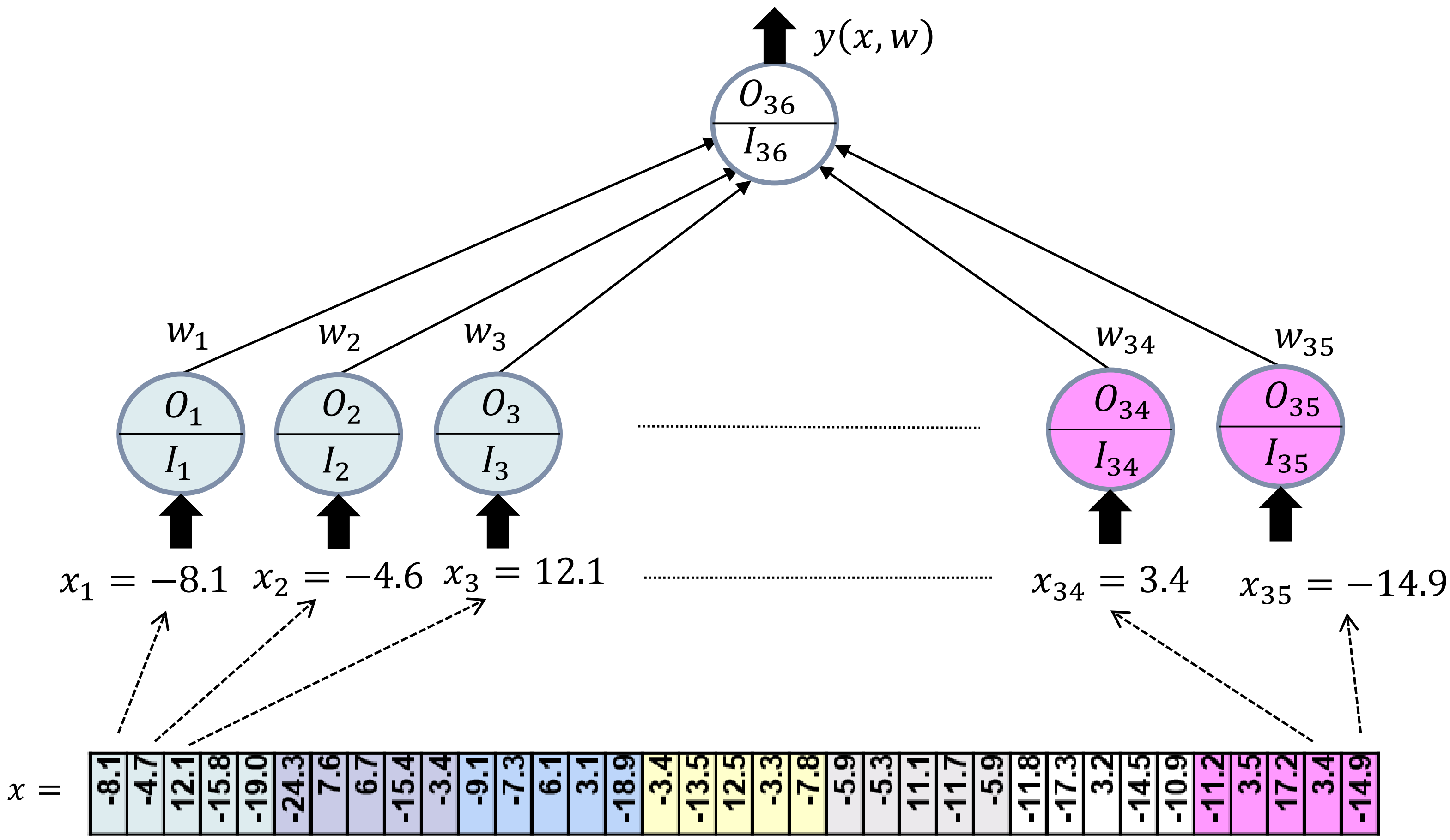
$\sigma_{rel} = 0.5$

-8.1
-4.7
12.1
-15.8
-19.0
-24.3
7.6
6.7
-15.4
-3.4
-9.1
-7.3
6.1
3.1
-18.9
-3.4
-13.5
12.5
-3.3
-7.8
-5.9
-5.3
11.1
-11.7
-5.9
-11.8
-17.3
3.2
-14.5
-10.9
-11.2
3.5
17.2
3.4
-14.9



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, number, 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 $[0.25, 1]$.
  - **`number`** 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: **this is the value to be used when generating the test dataset X^{TE}, y^{TE}.**

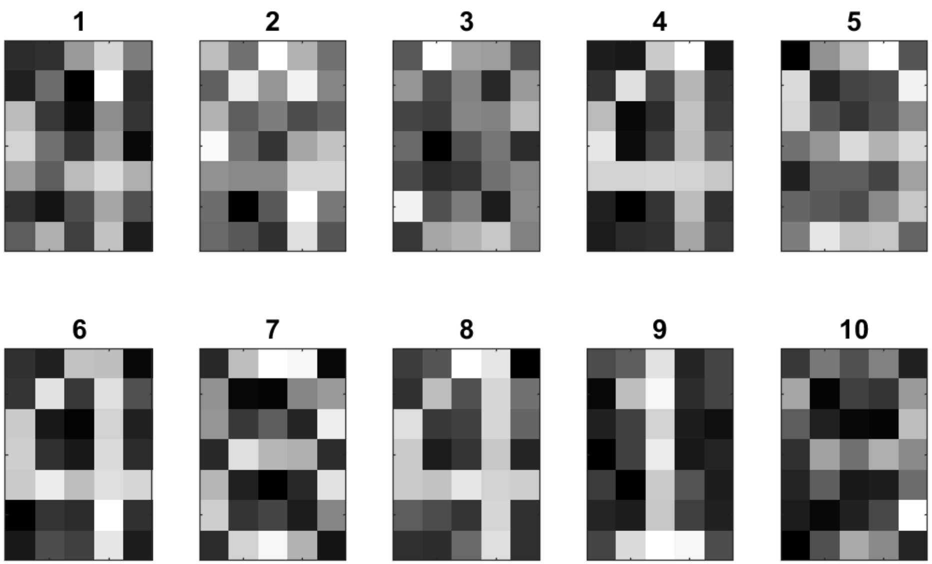
Training and test data set (2/2)

- **Exemple:** let `seed=1234`, `number=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);
```

- Defining the row vector of predictions $y(X^{TR},w)$ and the sigmoid matrix of inputs $\sigma(X^{TR})$ as

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

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

$$\tilde{L}(w;X^{TR},y^{TR},\lambda) = \frac{1}{p}\|y(X^{TR},w) - y^{TR}\|^2 + \lambda\frac{\|w\|^2}{2}$$

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

where $\circ$ stands for the **element-wise** (or **Hadamard**) **product** ($\begin{bmatrix}u\\v\end{bmatrix}\circ\begin{bmatrix}x\\y\end{bmatrix} = \begin{bmatrix}ux\\vy\end{bmatrix}$). These expressions can be easily coded in Matlab with the **element-wise operators " ./ " and " .* "**:

σ	`sig = @(Xds) 1./(1+exp(-Xds));`
y	`y   = @(Xds,w) sig(w'*sig(Xds));`
$\tilde{L}$	`L   = @(w,Xds,yds) (norm(y(Xds,w)-yds)^2)/size(yds,2)+ (la*norm(w)^2)/2;`
$\nabla\tilde{L}$	`gL  = @(w,Xds,yds) (2*sig(Xds)*((y(Xds,w)-yds).*y(Xds,w).* (1-y(Xds,w)))')/size(yds,2)+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 and QNM).
- **Part 2**: to extend code **`uo_nn_solve`** with the **stochastic gradient method (SGM)**.
- **Part 3**: to conduct a **comparation of the performance** of the abovementioned methods.

Part 1: pattern 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 =[1];
tr_freq    = .5;
tr_p       = 250;
te_q       = 250;
tr_seed    = 123456;
te_seed    = 789101;
%
% Parameters for optimization
%
la = 0.0;                                                       % 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.
sg_seed = 565544; sg_al0 = 2; sg_be = 0.3; sg_ga = 0.01;        % SGM iteration.
sg_emax = kmax; sg_ebest = floor(0.01*sg_emax);                 % SGM stopping condition.
%
% 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,sg_al0,sg_be,sg_ga,sg_emax,sg_ebest,sg_seed,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}) and the test dataset (X^{TE}, y^{TE}) with function `uo_nn_dataset.m`.
  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} and Accuracy^{TE}.

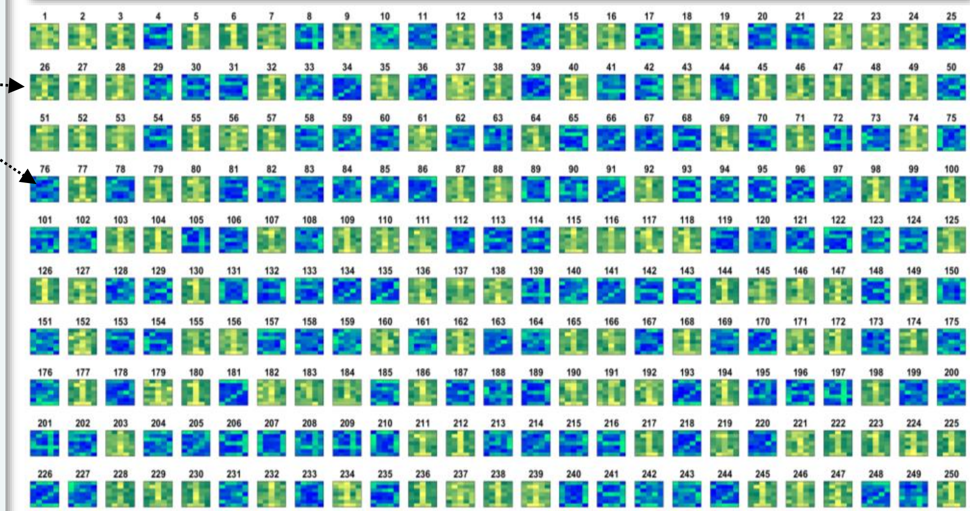
Part 1: example 1, `num_target=[1]`

```
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Pattern recognition with neural networks (OM/GCED).
[uo_nn_solve] 11-Feb-2021 19:34:47
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Training data set generation.
[uo_nn_solve]    num_target = 1
[uo_nn_solve]    tr_freq    = 0.50
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve]    tr_seed    = 123456
[uo_nn_solve] Test data set generation.
[uo_nn_solve]    te_freq    = 0.00
[uo_nn_solve]    te_q       = 250
[uo_nn_solve]    te_seed    = 789101
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.0000
[uo_nn_solve]    epsG= 1.0e-06, kmax= 10000
[uo_nn_solve]    ils= 3, ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn_solve]    c1= 0.01, c2= 0.45, isd= 1
[uo_nn_solve]    k        al  iW       g'*d         f       ||g||
[uo_nn_solve]     1   1.00e+00   2  -1.18e-01   2.50e-01   3.44e-01
[uo_nn_solve]     2   3.44e+00   0  -5.81e-02   1.50e-01   2.41e-01
[uo_nn_solve]     3   5.22e+01   0  -4.37e-03   3.59e-02   6.61e-02
.......................................................................
[uo_nn_solve]    42   3.09e+04   0  -9.68e-12   1.08e-06   3.11e-06
[uo_nn_solve]    43   2.18e+05   2  -1.47e-12   9.17e-07   1.21e-06
[uo_nn_solve]    44   6.95e+04   0  -1.75e-12   6.74e-07   1.32e-06
[uo_nn_solve]    45                             6.11e-07   9.80e-07
[uo_nn_solve]    k        al  iW       g'*d         f       ||g||
[uo_nn_solve]    wo=[
[uo_nn_solve]         -1.1e+00,-1.6e+00,-2.3e-01,-2.4e+00,-1.4e+00
[uo_nn_solve]         -1.3e+00,+1.5e+00,+2.7e+00,+1.3e-01,-1.4e+00
[uo_nn_solve]         -1.7e+00,-9.3e-02,+3.0e+00,-7.9e-01,-7.6e-01
[uo_nn_solve]         -2.1e+00,-1.7e+00,+2.7e+00,-2.4e+00,-1.4e-01
[uo_nn_solve]         -1.8e+00,-4.8e-01,+2.0e+00,-1.4e+00,-2.0e+00
[uo_nn_solve]         -9.6e-01,-1.2e+00,+2.2e+00,-1.1e+00,-3.6e-01
[uo_nn_solve]         -2.5e-01,-2.1e-02,+2.7e-01,+7.4e-01,-7.4e-01
[uo_nn_solve]       ]
[uo_nn_solve] Accuracy.
[uo_nn_solve] tr_accuracy = 100.0
[uo_nn_solve] te_accuracy = 100.0
wall time = 1.6e-01 s.
```

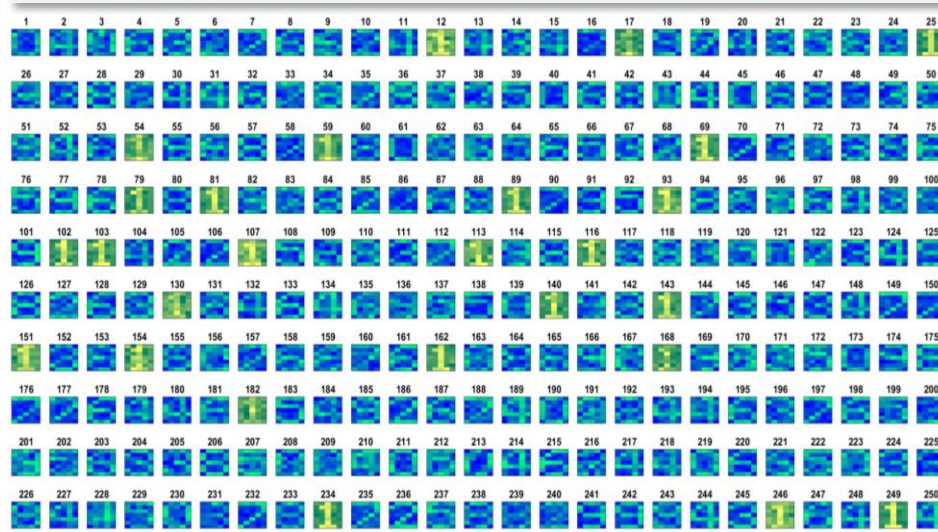
Rigth positive

Rigth negative

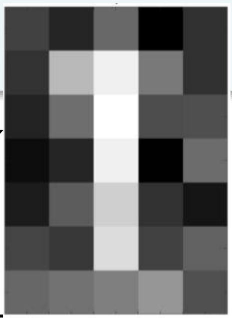
```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



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



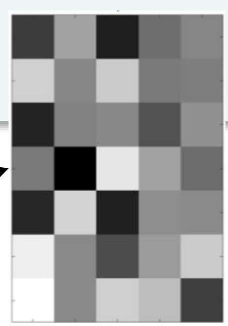
```
>> uo_nn_Xyplot(wo,0,[])
```



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

```
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Pattern recognition with neural networks (OM/GCED).
[uo_nn_solve] 11-Feb-2021 20:03:11
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Training data set generation.
[uo_nn_solve]    num_target = 3
[uo_nn_solve]    tr_freq    = 0.50
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve]    tr_seed    = 123456
[uo_nn_solve] Test data set generation.
[uo_nn_solve]    te_freq    = 0.00
[uo_nn_solve]    te_q       = 250
[uo_nn_solve]    te_seed    = 789101
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.0000
[uo_nn_solve]    epsG= 1.0e-06, kmax= 10000
[uo_nn_solve]    ils= 3, ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn_solve]    c1= 0.01, c2= 0.45, isd= 1
[uo_nn_solve]    k        al   iW      g'*d          f       ||g||
[uo_nn_solve]     1   1.00e+00   2  -3.92e-02   2.50e-01   1.98e-01
[uo_nn_solve]    23   1.41e+01   0  -7.56e-05   1.80e-02   8.70e-03
[uo_nn_solve]    45   1.07e+01   0  -2.36e-05   1.00e-02   4.86e-03
                 ..........................................................
[uo_nn_solve]  2069   9.54e+03   0  -2.50e-12   3.58e-06   1.58e-06
[uo_nn_solve]  2091   1.01e+04   0  -2.22e-12   3.33e-06   1.49e-06
[uo_nn_solve]  2113   1.07e+04   0  -1.97e-12   3.09e-06   1.40e-06
[uo_nn_solve]  2128                             2.94e-06   9.93e-07
[uo_nn_solve]    k        al   iW      g'*d          f       ||g||
[uo_nn_solve]    wo=[
[uo_nn_solve]         -8.1e+00,+1.5e+00,-1.1e+01,-3.4e+00,-1.2e+00
[uo_nn_solve]         +5.9e+00,-1.2e+00,+5.3e+00,-2.3e+00,-2.0e+00
[uo_nn_solve]         -1.0e+01,-1.6e+00,-1.0e+00,-6.1e+00,-1.2e-01
[uo_nn_solve]         -2.4e+00,-1.4e+01,+7.9e+00,+1.5e+00,-3.6e+00
[uo_nn_solve]         -1.0e+01,+6.0e+00,-1.1e+01,-4.3e-01,-6.9e-01
[uo_nn_solve]         +8.7e+00,-9.0e-01,-6.6e+00,+7.8e-01,+5.3e+00
[uo_nn_solve]         +1.0e+01,-7.7e-01,+5.6e+00,+4.0e+00,-7.8e+00
[uo_nn_solve]       ]
[uo_nn_solve] Accuracy.
[uo_nn_solve] tr_accuracy = 100.0
[uo_nn_solve] te_accuracy = 96.0
 wall time = 2.5e+00 s.
```

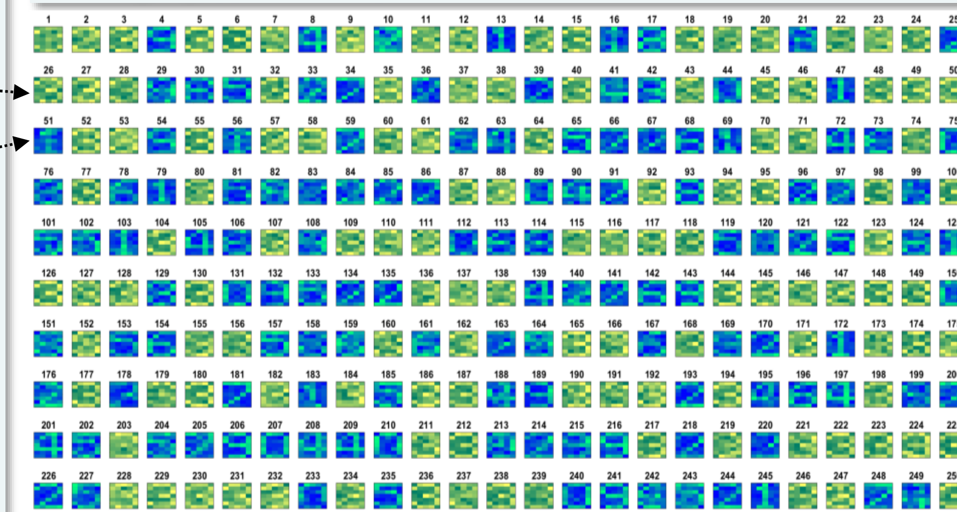
```
>> uo_nn_Xyplot(wo,0,[])
```



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

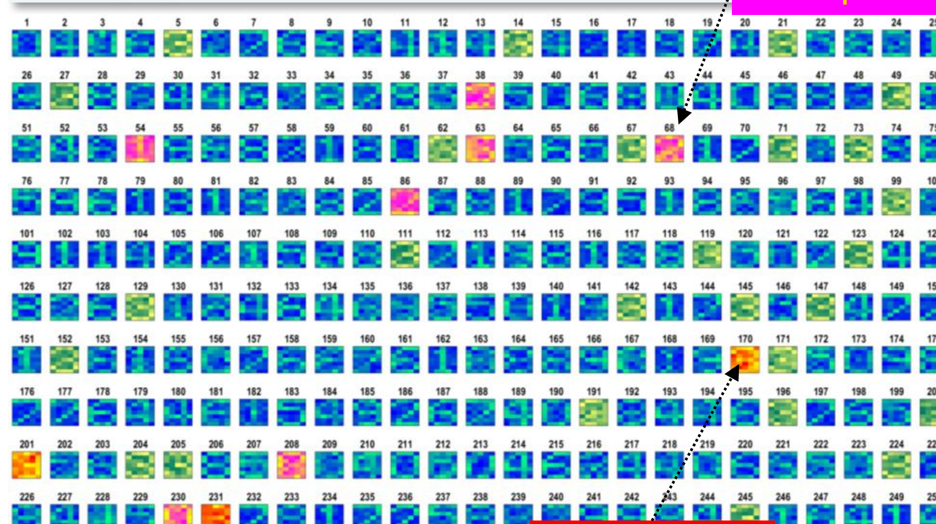
Rigth positive

Rigth negative



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

False positive

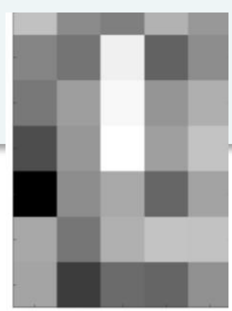


False negative

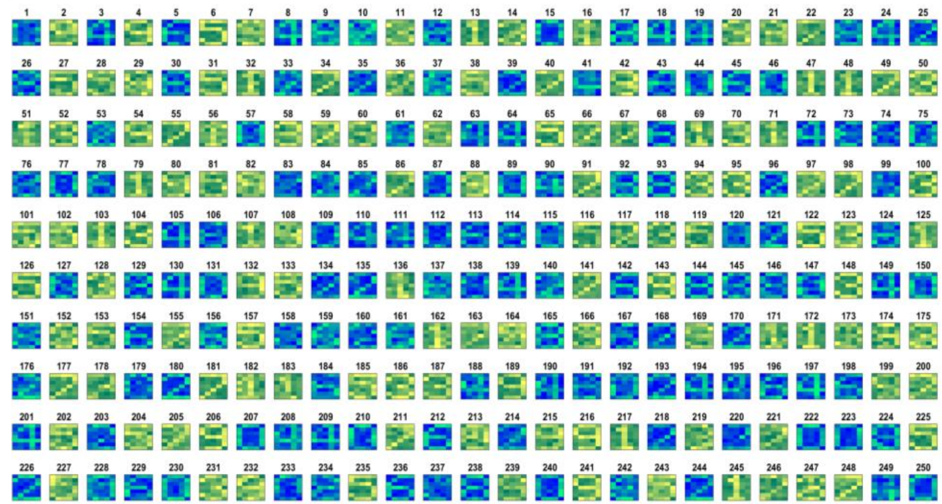
Part 1: example 3, `num_target=[1 3 5 7 9]`

```
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Pattern recognition with neural networks (OM/GCED).
[uo_nn_solve] 11-Feb-2021 20:14:05
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Training data set generation.
[uo_nn_solve]    num_target = 1 3 5 7
[uo_nn_solve]    tr_freq    = 0.50
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve]    tr_seed    = 123456
[uo_nn_solve] Test data set generation.
[uo_nn_solve]    te_freq    = 0.00
[uo_nn_solve]    te_q       = 250
[uo_nn_solve]    te_seed    = 789101
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.0000
[uo_nn_solve]    epsG= 1.0e-06, kmax= 10000
[uo_nn_solve]    ils= 3, ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn_solve]    c1= 0.01, c2= 0.45, isd= 1
[uo_nn_solve]    k        al  iW      g'*d          f       ||g||
[uo_nn_solve]     1   1.00e+00   2  -3.41e-02   2.50e-01   1.85e-01
[uo_nn_solve]    11   3.36e+00   0  -1.56e-03   4.89e-02   3.95e-02
.....................................................................
[uo_nn_solve]  1821   1.56e+04   0  -2.31e-12   4.02e-06   1.52e-06
[uo_nn_solve]  1841   1.49e+04   0  -1.90e-12   3.70e-06   1.38e-06
[uo_nn_solve]  1861   1.53e+04   0  -1.71e-12   3.43e-06   1.31e-06
[uo_nn_solve]  1881   1.55e+04   0  -1.57e-12   3.17e-06   1.25e-06
[uo_nn_solve]  1901   1.55e+04   0  -1.47e-12   2.94e-06   1.21e-06
[uo_nn_solve]  1914                           2.79e-06   9.90e-07
[uo_nn_solve]    k        al  iW      g'*d          f       ||g||
[uo_nn_solve]    wo=[
[uo_nn_solve]         +6.1e+00,-7.3e-01,-2.3e+00,+4.1e+00,+9.4e-01
[uo_nn_solve]         -1.5e+00,-3.6e+00,+1.2e+01,-6.0e+00,-4.5e-01
[uo_nn_solve]         -3.1e+00,+1.6e+00,+1.3e+01,+5.3e-01,+3.9e+00
[uo_nn_solve]         -8.5e+00,+8.0e-01,+1.4e+01,+1.8e+00,+6.4e+00
[uo_nn_solve]         -1.8e+01,-5.4e-01,+3.1e+00,-5.4e+00,+2.3e+00
[uo_nn_solve]         +2.9e+00,-3.4e+00,+4.0e+00,+6.4e+00,+6.3e+00
[uo_nn_solve]         +2.4e+00,-1.1e+01,-4.8e+00,-5.5e+00,+8.7e-02
[uo_nn_solve]       ]
[uo_nn_solve] Accuracy.
[uo_nn_solve] tr_accuracy = 100.0
[uo_nn_solve] te_accuracy = 95.6
 wall time = 2.2e+00 s.
```

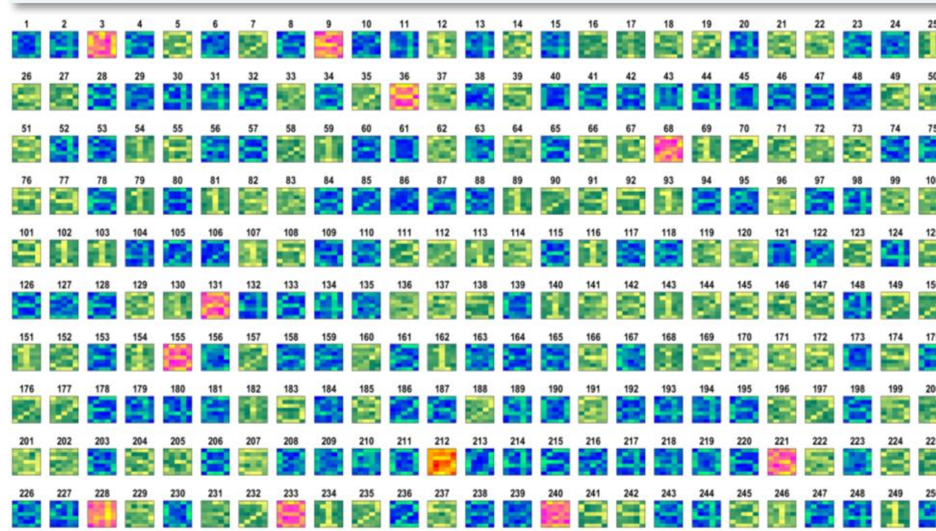
```
>> uo_nn_Xyplot(wo,0,[])
```



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



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



Part 2: Stochastic Gradient (1/6)

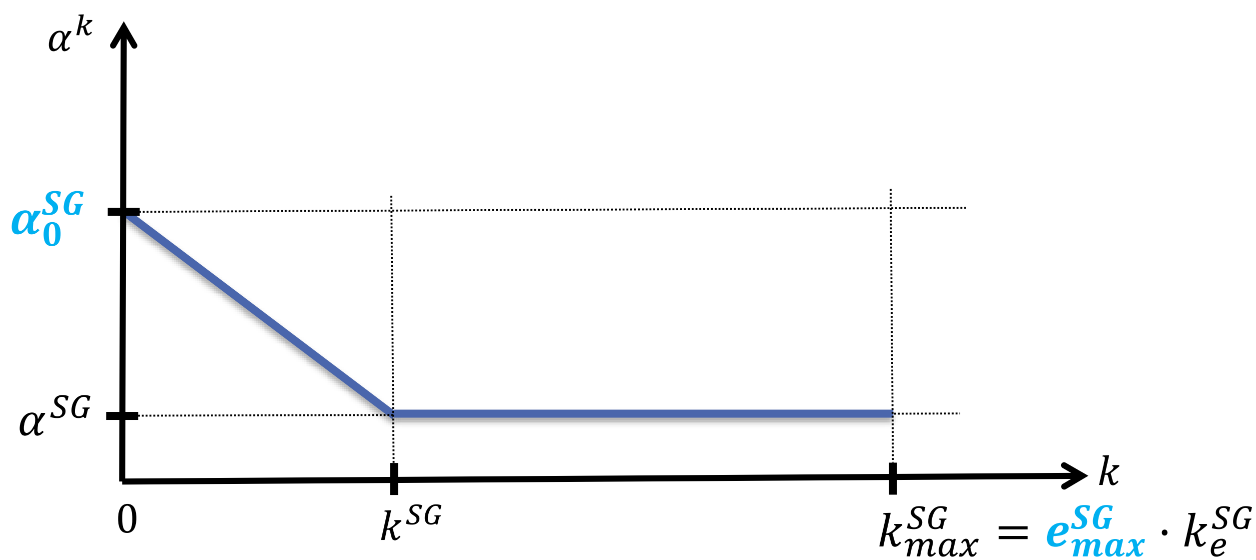
- **Stochastic Gradient Method (SGM).** (https://www.deeplearningbook.org/, chapter 8)
  - **Principle:** it is possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d. from the training dataset.
  - **Procedure:** given w^0 and $\boldsymbol{\alpha_0^{SG}} > 0$, $\boldsymbol{\beta^{SG}}$, $\boldsymbol{\gamma^{SG}} \in [0,1]$ and $\boldsymbol{e_{max}^{SG}} > 0$. $k \leftarrow 0$.

1. Take a random permutation of the training data set: $\boldsymbol{P} \leftarrow \{\boldsymbol{p_1}, \boldsymbol{p_2}, \dots, \boldsymbol{p_p}\}$
2. Define the size of the minibatch: $\boldsymbol{m} \coloneqq \lfloor \boldsymbol{\gamma^{SG}} \cdot p \rfloor$ ($\ll p$).
3. Define a minibatch of observations associated to the first $\boldsymbol{m}$ indexs of $\boldsymbol{P}$:
   - Indexes of the minibatch: $\boldsymbol{S} \leftarrow \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_m}\} \equiv \{\boldsymbol{p_1}, \boldsymbol{p_2}, \dots, \boldsymbol{p_m}\}$.
   - Observations of the minibatch: $\begin{cases} X_{\boldsymbol{S}}^{TR} \leftarrow \left[x_{\boldsymbol{s_1}}^{TR}, x_{\boldsymbol{s_2}}^{TR}, \dots, x_{\boldsymbol{s_m}}^{TR}\right] \\ y_{\boldsymbol{S}}^{TR} \leftarrow \left[y_{\boldsymbol{s_1}}^{TR} \quad y_{\boldsymbol{s_2}}^{TR} \quad \cdots \quad y_{\boldsymbol{s_m}}^{TR}\right]^T \end{cases}$
4. Estimate the gradient search direction : $d^k \leftarrow -\nabla \tilde{L}\left(w^k; X_{\boldsymbol{S}}^{TR}, y_{\boldsymbol{S}}^{TR}, \lambda\right)$
5. Update the parametres: $w^k \leftarrow w^k + \alpha^k d^k$ with ***learning rate $\boldsymbol{\alpha^k}$***. $k \leftarrow k + 1$
6. Repeat steps 3 to 5 $\boldsymbol{k_e^{SG}} := \lceil \boldsymbol{p}/\boldsymbol{m} \rceil$ iterations, taking at every iteration the next bunch of $\boldsymbol{m}$ indexes of $\boldsymbol{P}$ until every observation is used (***epoch***).
7. Generate a new permutation $\boldsymbol{P}$ and repeat the steps 2 to 6 for a number of $\boldsymbol{e_{max}^{SG}}$ epochs (a total of $\boldsymbol{k_{max}^{SG}} := \boldsymbol{e_{max}^{SG}} \cdot \boldsymbol{k_e^{SG}}$ iterations).

Part 2: Stochastic Gradient (2/6)

– **Learning rate $\boldsymbol{\alpha^k}$:** linear decay until iteration k^{SG} of a total of k^{SG}_{max} iterations, then constant α^{SG}

$$\alpha^k \leftarrow \begin{cases} \left(1-\dfrac{k}{k^{SG}}\right)\boldsymbol{\alpha_0^{SG}} + \dfrac{k}{k^{SG}}\alpha^{SG} & \text{if } k \le k^{SG} \\ \alpha^{SG} & \text{if } k > k^{SG} \end{cases}, \qquad \begin{cases} \alpha^{SG} := 0{,}01 \cdot \boldsymbol{\alpha_0^{SG}} \\ k^{SG} := \lfloor \boldsymbol{\beta^{SG}} \cdot k^{SG}_{max} \rfloor \end{cases}$$



Part 2: Stochastic Gradient (3/6)

– **Stopping criterion:** the usual stopping condition $\|\nabla f(x^k)\| \approx 0$ is no longer suitable, as the SGM does not compute the true gradient $\nabla f(x)$. One of the most commonly regularization stopping condition is the ***early stopping condition***. The rationale of this stopping criterion is:

1. To check the value of the loss function for the **test data set** after each epoch e:

$$\tilde{L}^{TE} \leftarrow \tilde{L}\left(w^{e \cdot k_e^{SG}}, \boldsymbol{X}^{\boldsymbol{TE}}, \boldsymbol{y}^{\boldsymbol{TE}}, \lambda\right).$$

2. Every time the new value $\tilde{L}^{TE}$ improves the best value so far ($\tilde{L}^{TE}_{best}$), $\boldsymbol{\tilde{L}^{TE}}$ **and the associated solution** $\boldsymbol{w^{e \cdot k_e^{SG}}}$ **is saved** :

$$\tilde{L}^{TE} < \tilde{L}^{TE}_{best} \stackrel{\text{def}}{=} \min_{j=1,\dots,e-1}\left\{\tilde{L}\left(w^{j \cdot k_e^{SG}}, \boldsymbol{X}^{\boldsymbol{TE}}, \boldsymbol{y}^{\boldsymbol{TE}}, \lambda\right)\right\} \Rightarrow \begin{cases} \boldsymbol{\tilde{L}^{TE}_{best}} \leftarrow \boldsymbol{\tilde{L}^{TE}} \\ \boldsymbol{w}^* \leftarrow \boldsymbol{w^{e \cdot k_e^{SG}}} \end{cases}$$

3. If the value of $\tilde{L}^{TE}_{best}$ has not been improved (updated) for the last $\boldsymbol{e^{SG}_{best}}$ epochs, the algorithm is terminated, and the solution of the last update w^* is declared optimal.

Part 2: Stochastic Gradient (4/6)

Algorithm SGM: Stochastic Gradient Method.

$w^* \leftarrow$ SGM($w^0, \lambda, \tilde{L}, \nabla\tilde{L}, X^{TR}, y^{TR}, X^{TE}, y^{TE}, \alpha_0^{SG}, \beta^{SG}, \gamma^{SG}, e_{max}^{SG}, e_{best}^{SG}$)

$p :=$ columns(X^{TR});

$m := \lfloor \gamma^{SG} \cdot p \rfloor$; $k_e^{SG} := \lceil p/m \rceil$; $k_{max}^{SG} := e_{max}^{SG} \cdot k_e^{SG}$; $e := 0$; $s := 0$; $\tilde{L}_{best}^{TE} := +\infty$; $k := 0$;

Epochs

While $e \le e_{max}^{SG}$ and $s < e_{best}^{SG}$ do

$P \leftarrow \{p_1, p_2, \dots, p_p\}$;

Minibatches

For $i = 0,1,\dots,\lceil p/m - 1 \rceil$ do

$S \leftarrow \{s_1, s_2, \dots, s_m\} \equiv \{p_{i\cdot m+1}, p_{i\cdot m+2}, \dots, p_{\min\{(i+1)\cdot m,\, p\}}\}$;

$X_S^{TR} \leftarrow [x_{s_1}^{TR}, x_{s_2}^{TR}, \dots, x_{s_m}^{TR}]$; $y_S^{TR} \leftarrow [y_{s_1}^{TR} \quad y_{s_2}^{TR} \quad \cdots \quad y_{s_m}^{TR}]^T$;

$d^k \leftarrow -\nabla\tilde{L}(w^k; X_S^{TR}, y_S^{TR}, \lambda)$;

$$\alpha^k \leftarrow \begin{cases} \left(1 - \dfrac{k}{k^{SG}}\right)\alpha_0^{SG} + \dfrac{k}{k^{SG}}\alpha^{SG} & \text{if } k \le k^{SG} \\ \alpha^{SG} & \text{if } k > k^{SG} \end{cases}, \qquad \begin{cases} \alpha^{SG} := 0{,}01 \cdot \alpha_0^{SG} \\ k^{SG} := \lfloor \beta^{SG} \cdot k_{max}^{SG} \rfloor \end{cases}$$

$w^k \leftarrow w^k + \alpha^k d^k$; $k \leftarrow k + 1$;

End For

$e \leftarrow e + 1$; $\tilde{L}^{TE} \leftarrow \tilde{L}(w^k, X^{TE}, y^{TE}, \lambda)$;

If $\tilde{L}^{TE} < \tilde{L}_{best}^{TE}$ then ($\tilde{L}_{best}^{TE} \leftarrow \tilde{L}^{TE}$, $w^* \leftarrow w^k$, $s \leftarrow 0$) else $s \leftarrow s + 1$ End If

End While

End SGM

Part 2: Stochastic Gradient (5/6)

- Script `uo_nn_solve.m` with `isd = 7` now applies the SGM:

uo_nn_main.m : recognition of num_target digits.

```
clear;
%
% Parameters for dataset generation
%
num_target =[1];
tr_freq    = .5;
tr_p       = 250;
te_q       = 250;
tr_seed    = 123456;
te_seed    = 789101;
%
% Parameters for optimization
%
la = 0.0;                                                  % 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 = 7; icg = 2; irc = 2 ; nu = 1.0;                      % Search direction.
sg_seed = 565544; sg_al0 = 2; sg_be = 0.3; sg_ga = 0.01;   % SGM iteration.
sg_emax = kmax; sg_ebest = floor(0.01*sg_emax);            % SGM stopping condition.
%
% Optimization
%
t1=clock;
global iheader; iheader = 1;
[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,sg_al0,sg_be,sg_ga,sg_emax,sg_ebest,sg_seed,icg,irc,nu);
t2=clock;
fprintf(' wall time = %6.1d s.\n', etime(t2,t1));
```

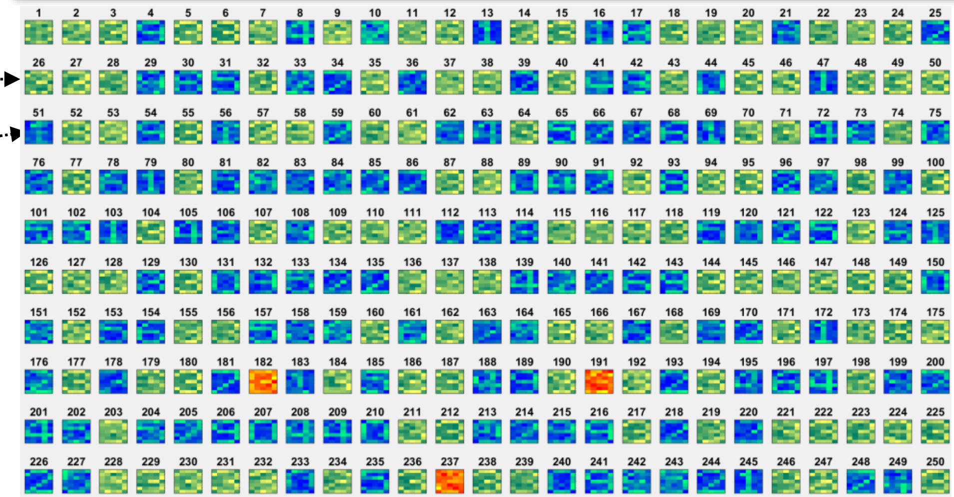
Part 2: Stochastic Gradient (6/6)

```
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Pattern recognition with neural networks (OM/GCED).
[uo_nn_solve] 26-Apr-2021 16:45:24
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Training data set generation.
[uo_nn_solve]    num_target = 3
[uo_nn_solve]    tr_freq    = 0.50
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve]    tr_seed    = 123456
[uo_nn_solve] Test data set generation.
[uo_nn_solve]    te_freq    = 0.00
[uo_nn_solve]    te_q       = 250
[uo_nn_solve]    te_seed    = 789101
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.0000
[uo_nn_solve]    epsG= 1.0e-06, kmax= 10000
[uo_nn_solve]    ils= 3, ialmax= 2, kmaxBLS= 30, epsBLS= 1.0e-03,
[uo_nn_solve]    c1= 0.01, c2= 0.45, isd= 7
[uo_nn_solve]    sg_al0= 2.00, sg_be= 0.3, sg_ge= 0.01
[uo_nn_solve]    sg_emax= 10000, sg_ebest= 100
[uo_nn_solve]    k          al   iW        g'*d          f         ||g||
[uo_nn_solve]      1    2.00e+00    0   -8.34e-02    2.50e-01    1.98e-01
[uo_nn_solve]    129    2.00e+00    0   -1.25e-03    3.43e-02    8.60e-02
[uo_nn_solve]    257    2.00e+00    0   +1.07e-05    1.12e-02    1.41e-02
[uo_nn_solve]    385    2.00e+00    0   +1.91e-07    7.41e-03    3.10e-03
[uo_nn_solve]    513    2.00e+00    0   -4.22e-05    7.31e-03    1.08e-02
[uo_nn_solve]    641    2.00e+00    0   -1.34e-04    1.07e-02    1.99e-02
          ...      ..........................................................
[uo_nn_solve] 12673    1.93e+00    0   -2.21e-08    3.56e-04    1.78e-04
[uo_nn_solve] 12751                                 4.40e-04    1.60e-03
[uo_nn_solve]    k          al   iW        g'*d          f         ||g||
[uo_nn_solve]    wo=[
[uo_nn_solve]         -5.6e-01,-1.7e-02,-1.3e+00,-2.8e-01,-1.2e+00
[uo_nn_solve]         +9.3e-01,-1.3e-01,-2.1e-01,-9.6e-03,+3.3e-01
[uo_nn_solve]         -2.8e+00,-6.4e-01,-6.1e-01,-8.9e-01,-1.8e-01
[uo_nn_solve]         -6.2e-01,-3.1e+00,+1.6e+00,+5.7e-01,-1.8e+00
[uo_nn_solve]         -2.1e+00,+4.4e-01,-2.1e+00,-2.5e-01,+2.6e-01
[uo_nn_solve]         +2.4e+00,-9.4e-01,-4.9e-01,+2.4e-01,+7.6e-01
[uo_nn_solve]         -5.0e-01,-3.6e-01,+2.1e-01,+2.8e-01,-6.9e-01
[uo_nn_solve]       ]
[uo_nn_solve] Accuracy.
[uo_nn_solve] tr_accuracy = 98.8
[uo_nn_solve] te_accuracy = 98.8
 wall time = 2.8e-01 s.
```

Rigth positive

Rigth negative

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

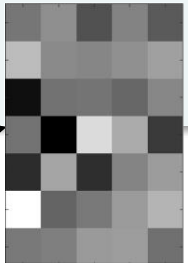


```
>> uo_nn_Xyplot(Xte,yte,wo)
```

False positive

False negative

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



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 $\boldsymbol{\lambda} \in \{\mathbf{0.0}, \mathbf{0.01}, \mathbf{0.1}\}$.
  - For every optimization algorithm: **GM, QNM** and **SGM**.

  That makes a total of $\mathbf{10} \times \mathbf{3} \times \mathbf{3} = \mathbf{90}$ **instances to be solved**.
- We will use a small size dataset with

```
tr_p = 250; te_q = 250; tr_freq = 0.5;
```

- Max. number of iterations for GM and QNM will be `kmax = 1000`. To do a fair comparison with SGM, `sg_emax = kmax`, so that every method is allowed to spend a similiar computational effort (number of gradient evaluations).

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.

```
clear;
% Parameters.
tr_seed = 123456; te_seed = 789101;
tr_p = 250; te_q = 250; tr_freq = 0.5;                      % Datasets generation
epsG = 10^-6; kmax = 1000;                                  % Stopping condition.
ils=3; ialmax = 1; kmaxBLS=10; epsal=10^-3; c1=0.01; c2=0.45; % Linesearch.
icg = 2; irc = 2 ; nu = 1.0;                                % Search direction.
sg_seed = 565544; sg_al0 = 2; sg_be = 0.3; sg_ga = 0.01;    % SGM iteration.
sg_emax = kmax; sg_ebest = floor(0.01*sg_emax);             % SGM stopping condition.
% Optimization
global iheader; 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, 0.01, 0.1]
        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,sg_al0,sg_be,sg_ga,sg_emax,sg_ebest,sg_seed,icg,irc,nu);
            if iheader == 1
                fprintf(fileID,'num_target;      la; isd;  niter;     tex; tr_acc; te_acc;        L*;\n');
            end
            fprintf(fileID,'          %1i; %7.4f;    %1i; %6i; %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);
```

Part 3: computational study (3/3)

- The outcome of **`uo_nn_batch.m`** is the **log 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;  0.0000;   1;      75;  0.1055;  100.0;  100.0;  5.72e-07;
          1;  0.0000;   3;       6;  0.0099;  100.0;  100.0;  2.91e-50;
          1;  0.0000;   7;  125001;  1.5164;  100.0;  100.0;  1.41e-05;
          1;  0.0100;   1;      58;  0.0955;  100.0;  100.0;  2.76e-02;
          1;  0.0100;   3;      45;  0.1026;  100.0;  100.0;  2.76e-02;
          1;  0.0100;   7;    3751;  0.0464;   96.4;   99.6;  6.74e-02;
          ...............................................................
          0;  0.1000;   1;      36;  0.0645;   99.6;   98.4;  1.45e-01;
          0;  0.1000;   3;      20;  0.0520;   99.6;   98.4;  1.45e-01;
          0;  0.1000;   7;    2751;  0.0336;   53.6;   92.4;  6.37e-01;
```

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)

1) **Study of the convergence:** 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**: compare the value at the optimal solution of the loss function $\tilde{L}$ for every combination λ-algorithm. According to the information in the log file Are the three algorithms able to identify a stationary solution of the loss function $\tilde{L}$?

   b) **Local convergence**:

      i. Compare the speed of convergence of the three algorithms in terms of the execution time and number of iterations.

      ii. Analyse how the speed of convergence of the three algorithms depend on the value of λ and try to find an explanation for the observed dependence, if any.

      iii. 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 in terms of local and global convergence and justify which is the most efficient for the minimization of $\tilde{L}$.

Report (2/3)

2) **Study of the recognition accuracy**: now, we are going to analyse the recognition accuracy of the SLNN, $Accuracy^{TE}$ (**`te_acc`**) for the different algorithmic options with a more realistic dataset with:

```
tr_p = 20000; te_q = tr_p/10; tr_freq = 0.0;
```

Run the training process for the ten digits with these parameters and every algorithm, GM, QNM, SGM, using, for each one of them, the value of λ showing the best $Accuracy^{TE}$ in the results of the previous section 1- a) with the smaller dataset (**`tr_p = 250`**).

a) Analyse the results. Based on this analyse, tell if there is a method that clearly outperforms the others in terms of speed and recognition accuracy.

b) Does the best combination λ-algorithm with respect to the maximization of $Accuracy^{TE}$ coincide with best combination with respect to the minimization of $\tilde{L}$ observed in the previous study 1-c)? Discuss the reasons for the discrepancy, in case the combinations do not coincide.

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

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`**, **`te_seed`** and **`sg_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.
- The mark will take into account formal quality, understandability and reading easiness, as long as the evidences provided to support the results of the analysis performed in the project (numerical values, tables, plots, …).

Grading criteria

1) Study of the convergence:	**Points**
a) Global convergence.	1,5
b) Local convergence.	2,5
c) Discussion.	1,0
2) Study of the accuracy for large porblems.	
a) Best method for large problems.	2,0
b) Discussion $Accuracy^{TE}$-$\tilde{L}$	2,0
3) Overall quality of the report.	1,0