Unconstrained optimization

GECD / MDS / MIRI - BarcelonaTech

Lab Assignment

Pattern recognition with Single Layer Neural Network (SLNN)

v5.0-03/25

F.-Javier Heredia

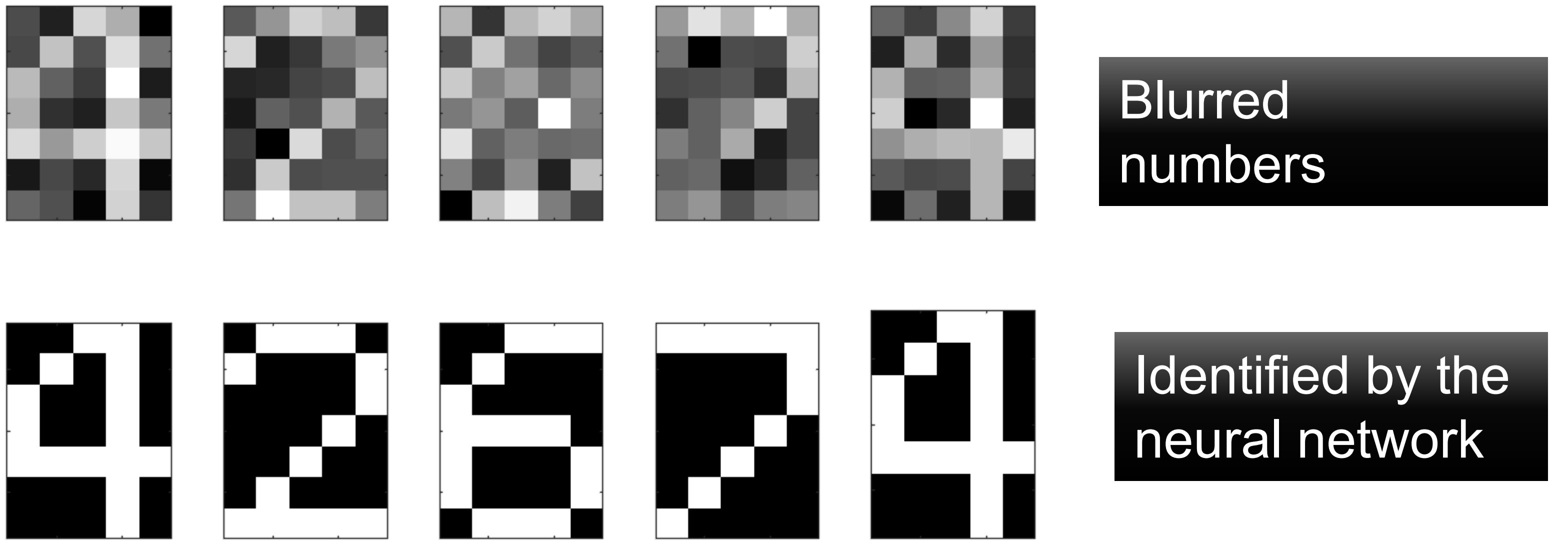
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

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:



Sequence=42674 ; Identified=42674

- To this end, we will formulate a **Single Layer Neural Network** that is going to be **trained to recognize** the different numbers with **First Derivative Optimization methods**.

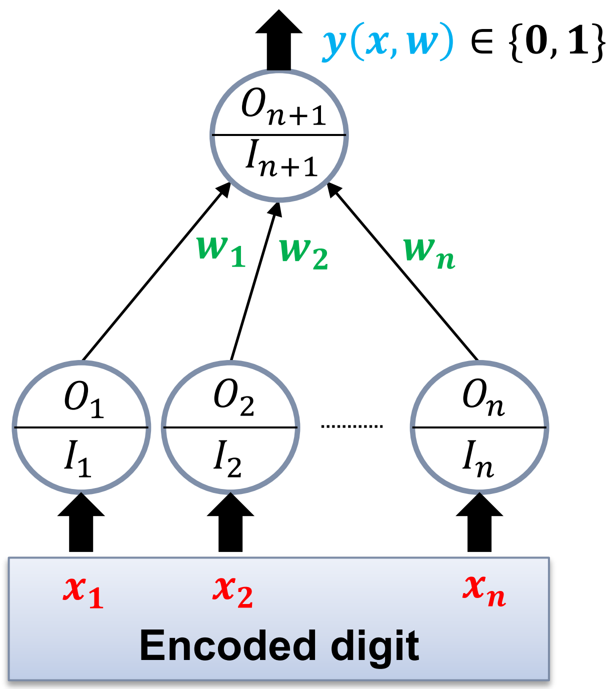
Supporting material

Documents
File `uo_nn_v50.pdf`: this document.

Functions/scripts	**Page**
Matlab's coding of the loss function and its gradient.	link
File `uo_nn_batch_st.m`: template to run a batch of recognitions.	link
File `uo_nn_solve_st.m`: template to run a single training and recognition.	link
File `uo_nn_solve.p`: executable for a single training and recognition, restricted.	link
File `uo_nn_dataset.p`: executable for the dataset generation.	link
File `uo_nn_Xyplot.p`: executable for the visualization of the dataset and results.	link

- All this material can be downloaded from `uonn.zip`.
- The executable `uo_nn_solve.p` needs the last version of library `uosol.zip`.

Single layer Neural Network (SLNN): architecture



- **Input signal (encoded digit):**

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

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

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

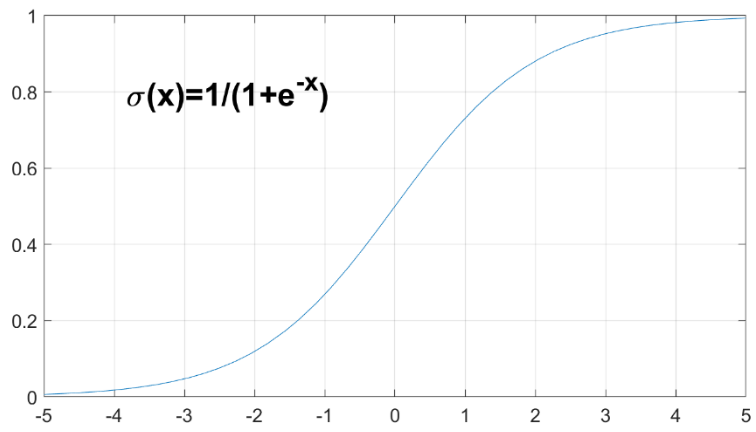
- **Output signal (recognition):**

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

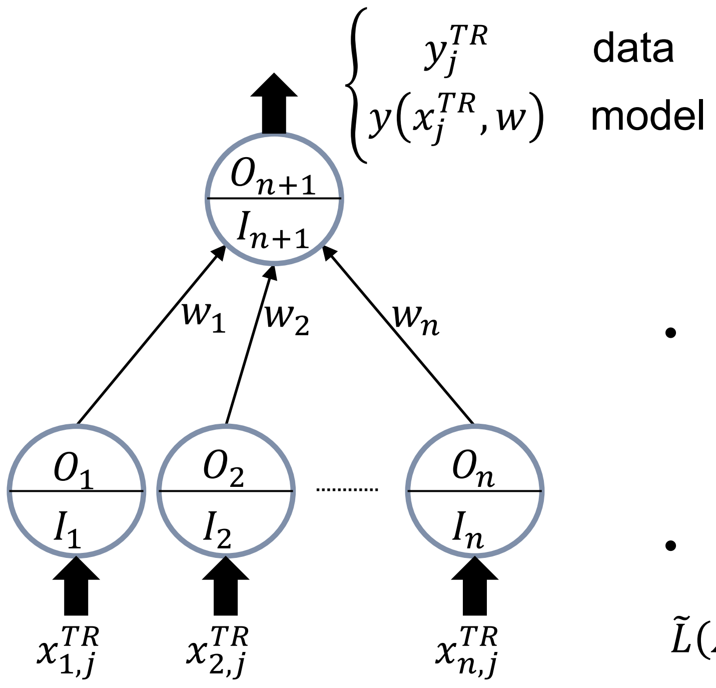
$$= \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)}\right)^{-1}$$

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

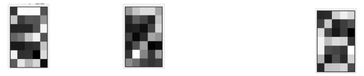
$y(x,w)$ should be $\begin{cases} = 0 & \text{if } x \neq \text{target digit} \\ = 1 & \text{if } x = \text{target digit} \end{cases}$



SLNN: training



- **Training dataset, size p:**



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

$$y^{TR} = \begin{bmatrix} y_1^{TR} & y_2^{TR} & \cdots & y_p^{TR} \end{bmatrix}^T = [1, 0, \dots, 0]^T \text{ (if target=3)}$$

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

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

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

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

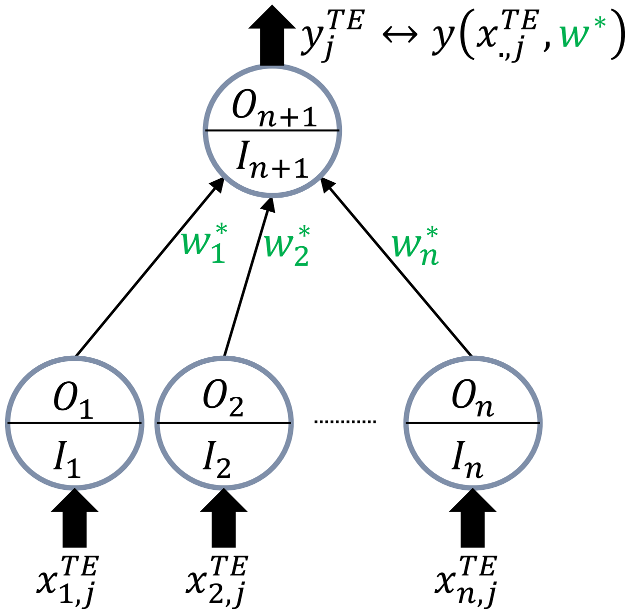
(1): AKA *ridge regression* or *Tikhonov regularization*.

- **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 dataset, size q:**

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

$$y^{TE} = \left[y_1^{TE} \quad y_2^{TE} \quad \cdots \quad y_q^{TE}\right]^T$$

- **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):**

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

- **Gradient:**

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

with

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

SLNN : gradient (2/2)

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

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

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

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

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

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 with fixed reduction ρ used so far cannot handle conveniently the SLNN problem. As a consequence 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):

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

  2. We will use function **`uoBLSNW32`**, the implementation of Alg 3.2 and 3.3 of N&W of a BLS based on interpolations (see N&W 3.4). This function is included in the library uolib.zip and can be activated with **`par.iAC=4`**):

  ```
  function [alpha,iout] = uoBLSNW32(f,g,x,d,almax,c1,c2)
  ```

  where

  ❖ **`f,g,d,x,almax,c1,c2`** are as usual (we propose **`c1=0.01`** and **`c2=0.9`**).

  ❖ **`iout=0`** if **`alpha`** satisfies SWC, **`iout=1`** if more than 30 iterations are done and **`iout=2`** if $\left|\alpha^{k+1} - \alpha^k\right| < 10^{-3}$.

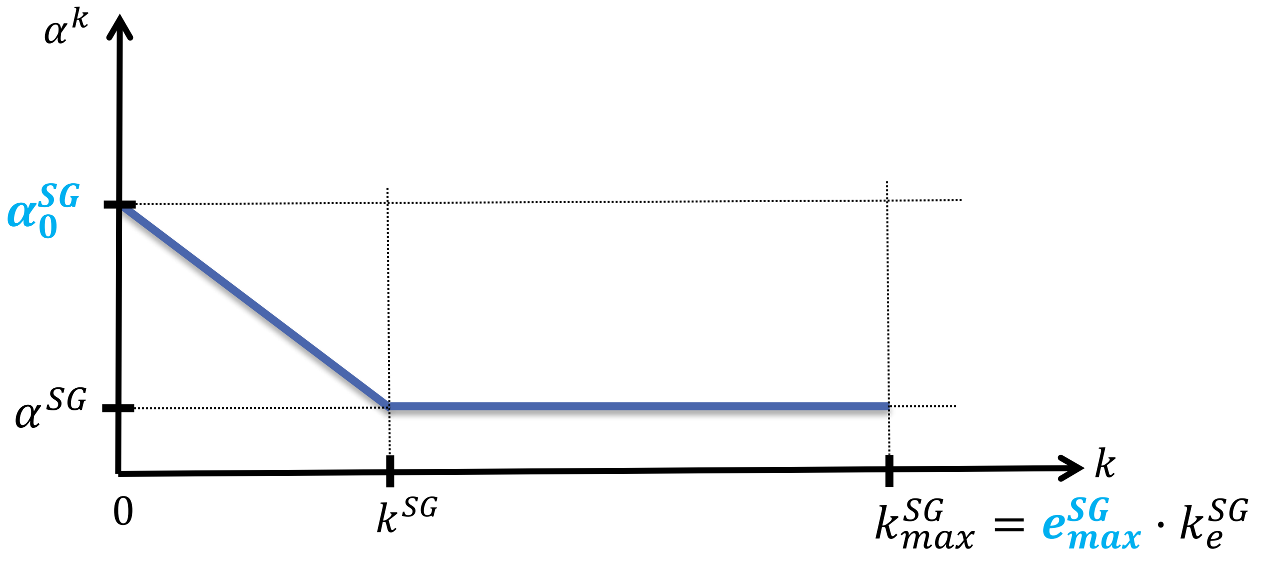
Stochastic Gradient (1/5)

- **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 $\boldsymbol{\alpha_0^{SG}} > 0$, $\boldsymbol{\beta^{SG}}$, $\boldsymbol{m^{SG}} \in [0,1] \ll p$ and $\boldsymbol{e_{max}^{SG}} > 0$. $k \leftarrow 0$.
    1. Take a random permutation of the training dataset: $\boldsymbol{P} \leftarrow \{\boldsymbol{p_1, p_2, \dots, p_p}\}$
    2. Define the size of the minibatch: $\boldsymbol{m} := \boldsymbol{m^{SG}}$.
    3. Define a minibatch of observations associated to the first $\boldsymbol{m}$ indexs of $\boldsymbol{P}$:
       - Indexes of the minibatch: $\boldsymbol{S} \leftarrow \{\boldsymbol{s_1, s_2, \dots, s_m}\} \equiv \{\boldsymbol{p_1, p_2, \dots, p_m}\}$.
       - Observations of the minibatch: $\begin{cases} X_{\boldsymbol{S}}^{TR} \leftarrow [x_{\boldsymbol{s_1}}^{TR}, x_{\boldsymbol{s_2}}^{TR}, \dots, x_{\boldsymbol{s_m}}^{TR}] \\ y_{\boldsymbol{S}}^{TR} \leftarrow [y_{\boldsymbol{s_1}}^{TR} \quad y_{\boldsymbol{s_2}}^{TR} \quad \cdots \quad y_{\boldsymbol{s_m}}^{TR}]^T \end{cases}$
    4. Estimate the gradient search direction : $d^k \leftarrow -\nabla\tilde{L}(w^k; X_{\boldsymbol{S}}^{TR}, y_{\boldsymbol{S}}^{TR}, \lambda)$
    5. Update the parametres: $w^k \leftarrow w^k + \alpha^k d^k$ with ***learning rate α^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 (the total number of iterations being $\boldsymbol{k_{max}^{SG}} := \boldsymbol{e_{max}^{SG}} \cdot \boldsymbol{k_e^{SG}}$).

Stochastic Gradient (2/5)

– **Learning rate α^k:** linear decay until iteration k^{SG} and then a constant value till the last iteration k_{max}^{SG}:

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



Stochastic Gradient (3/5)

– **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 dataset** 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 ($\tilde{L}^{TE}_{best}$), **$\tilde{L}^{TE}$ and the associated solution $w^{e\cdot k_e^{SG}}$ is saved** :

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

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

Stochastic Gradient (4/5)

Algorithm SGM: Stochastic Gradient Method.

```
$w^* \leftarrow$ SGM($w^0$,$\lambda$,$\tilde{L}$,$\nabla\tilde{L}$,$X^{TR}$,$y^{TR}$,$X^{TE}$,$y^{TE}$,$\alpha_0^{SG}$,$\beta^{SG}$,$m^{SG}$,$e_{max}^{SG}$,$e_{worse}^{SG}$)
        $p :=$ columns($X^{TR}$);
        $m := m^{SG}$; $k_e^{SG} := \lceil p/m \rceil$; $k_{max}^{SG} := e_{max}^{SG} \cdot k_e^{SG}$; $e := 0$; $s := 0$; $\tilde{L}_{best}^{TE} := +\infty$; $k := 0$;
Epochs
        While $e \le e_{max}^{SG}$ and $s < e_{worse}^{SG}$ do
            $P \leftarrow \{p_1, p_2, \dots, p_p\}$;
Minibatches
            For $i = 0,1,\dots,\lceil p/m - 1 \rceil$ do
                $S \leftarrow \{s_1, s_2, \dots, s_m\} \equiv \{p_{i\cdot m+1}, p_{i\cdot m+2}, \dots, p_{\min\{(i+1)\cdot m,\, p\}}\}$;
                $X_S^{TR} \leftarrow [x_{s_1}^{TR}, x_{s_2}^{TR}, \dots, x_{s_m}^{TR}]$; $y_S^{TR} \leftarrow [y_{s_1}^{TR} \quad y_{s_2}^{TR} \quad \cdots \quad y_{s_m}^{TR}]^T$;
                $d^k \leftarrow -\nabla\tilde{L}(w^k; X_S^{TR}, y_S^{TR}, \lambda)$;
                $\alpha^k \leftarrow \begin{cases} \left(1 - \frac{k}{k^{SG}}\right)\alpha_0^{SG} + \frac{k}{k^{SG}}\alpha^{SG} & \text{if } k \le k^{SG} \\ \alpha^{SG} & \text{if } k > k^{SG} \end{cases}$ , $\begin{cases} \alpha^{SG} := 0{,}01 \cdot \alpha_0^{SG} \\ k^{SG} := \lfloor \beta^{SG} \cdot k_{max}^{SG} \rfloor \end{cases}$
                $w^{k+1} \leftarrow w^k + \alpha^k d^k$; $k \leftarrow k + 1$;
            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
```

Stochastic Gradient (5/5)

```
[uo_nn_solve] Training data set generation.
[uo_nn_solve]    num_target = 10
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.1000
[uo_nn_solve]    w1= [0]
[uo_nn_solve]    Call uosol.
[uosol]
      isd= 7 (SGM)
      sg.seed = 565544, sg.m= 2
      sg.emax = 100, sg.eworse= 5
      sg.al0  = 2.00, sg.be= 0.30
      k         al          f
      1 +2.00e+00 +2.5e-01
      2 +2.00e+00 +6.8e-01
      3 +2.00e+00 +5.9e-01
      4 +2.00e+00 +4.8e-01
      5 +2.00e+00 +2.0e-01
   2116 +8.83e-01 +1.6e-01
   2117 +8.82e-01 +2.5e-01
   2118 +8.82e-01 +2.8e-01
   2119 +8.81e-01 +1.5e-01
   2120 +8.81e-01 +1.7e-01
   2121 +8.80e-01 +3.0e-01
   2122 +8.80e-01 +2.0e-01
   2123 +8.79e-01 +1.6e-01
   2124 +8.79e-01 +1.6e-01
   2125 +8.78e-01 +1.7e-01
   2126           +1.7e-01
      k         al          f
      ebest= 12, etot= 17,
      kbest= 1501, ktot= 2126
```

Training dataset with $p=$**tr_p m = 250**

$\lambda = 0.1, \qquad w^1 = [0]$

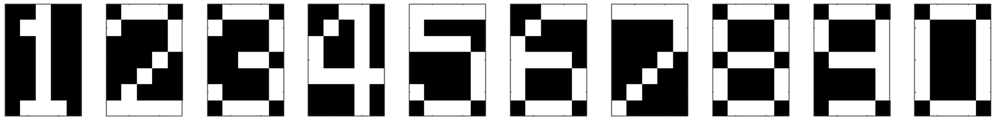
m^{SG} = **sg.m = 2**, α_0^{SG} = **sg.al0=2**, β^{SG} = **sg.be = 0.3**.
e_{max}^{SG} = **sg.emax = 100**, e_{worse}^{SG} = **sg.eworse = 5**.

Minibatch iterations

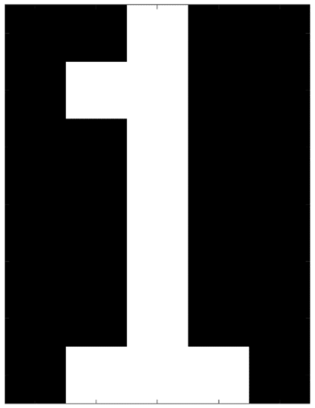
- A total of **etot = 17** epochs "e" has been done.
- A total of **ktot = 2126** mini-batches iterations "i" has been done.
- w^* is the iterate at the end of epoch **ebest= 12.**
- w^* is the iterate at the mini-batch iteration **kbest= 1501**

Dataset generation (1/4)

- 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

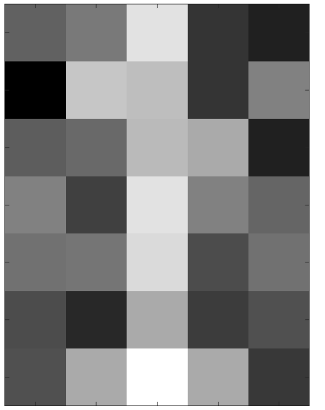
$$x = (-10, -10, 10, -10, -10, -10, 10, 10, -10, -10, -10, -10, 10, -10, -10, -10, -10, 10, -10, -10, -10, -10, 10, -10, -10, -10, -10, 10, -10, -10, -10, 10, 10, 10, -10)^T$$

Gaussian blur

$$x \leftarrow x + \epsilon = (-8.1, -4.7, 12.1, -15.8, -19.0, -24.3, 7.6, 6.7, -15.4, -3.4, -9.1, -7.3, 6.1, 3.1, -18.9, -3.4, -13.5, 12.5, -3.3, -7.8, -5.9, -5.3, 11.1, -11.7, -5.9, -11.8, -17.3, 3.2, -14.5, -10.9, -11.2, 3.5, 17.2, 3.4, -14.9)^T$$

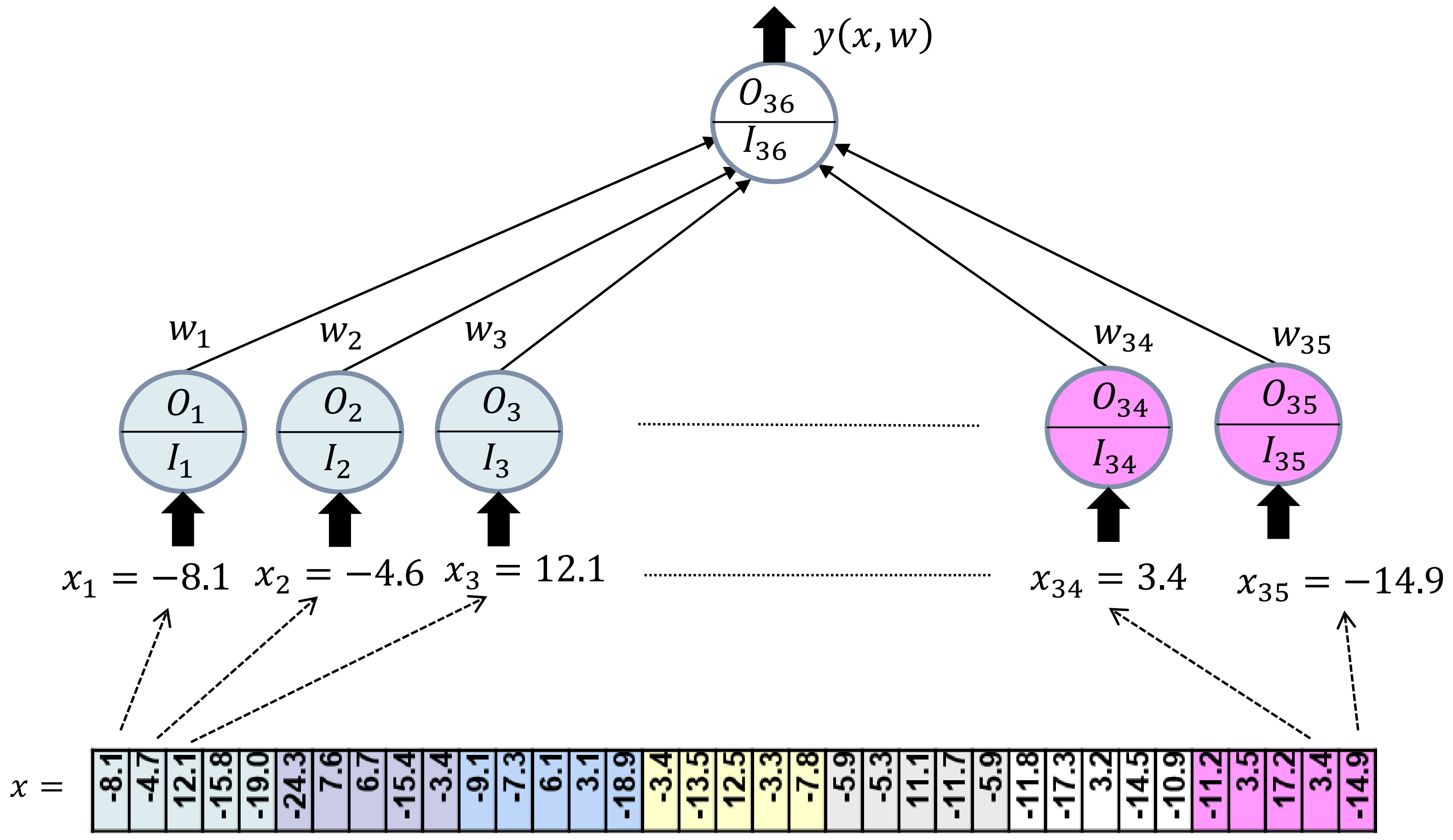
$\epsilon \sim N(0,5)$

$\sigma_{rel} = 0.5$



Dataset generation (2/4)

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



Dataset generation (3/4)

- 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 datasets:

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

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

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 datasets (**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 (one or several).
- **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 dataset **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}.**

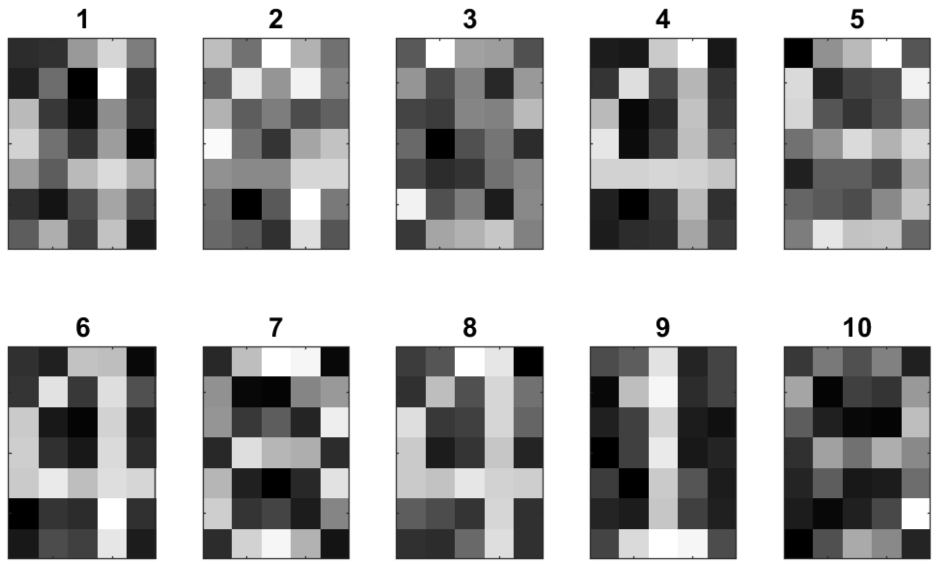
Dataset generation (4/4)

- **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).

Coding the loss function and gradient

- Let **Xtr,ytr** be the training dataset:

```
[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

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

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** ($\begin{bmatrix}u\\v\end{bmatrix}\circ\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}ux\\vy\end{bmatrix}$).

The Matlab code using the **element-wise operators “./ “ and “.* “** is:

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

Training, recognition and accuracies

```
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Pattern recognition with neural networks (OM/GCED).
[uo_nn_solve] 12-Jan-2025 13:32:55
[uo_nn_solve] ::::::::::::::::::::::::::::::::::::::::::::::::::
[uo_nn_solve] Training dataset generation.
[uo_nn_solve]    num_target = 8
[uo_nn_solve]    tr_freq    = 0.50
[uo_nn_solve]    tr_p       = 250
[uo_nn_solve]    tr_seed    = 63256
[uo_nn_solve] Test dataset generation.
[uo_nn_solve]    te_freq    = 0.00
[uo_nn_solve]    te_q       = 250
[uo_nn_solve]    te_seed    = 13625
[uo_nn_solve] Optimization
[uo_nn_solve]    L2 reg. lambda = 0.0000
[uo_nn_solve]    epsG= 1.0e-03, kmax=  100
[uo_nn_solve]    c1= 0.01, c2= 0.90, isd= 3
[uo_nn_solve]    w0= [0]
[uo_nn_solve]     k      alm       al      g'*d        f     ||g||
[uo_nn_solve]     1 1.00e+00 4.72e-01 -1.11e-01 2.50e-01 3.34e-01
[uo_nn_solve]     2 1.65e+00 8.30e-02 -4.56e-02 2.12e-01 1.72e-01
[uo_nn_solve]     3 7.01e-01 4.01e-01 -8.90e-02 1.81e-01 1.41e-01
[uo_nn_solve]     4 6.45e-01 2.68e-01 -9.66e-02 1.50e-01 1.36e-01
[uo_nn_solve]     5 6.36e-01 2.89e-01 -7.19e-02 1.27e-01 1.20e-01
                 ..............................................
[uo_nn_solve]    57 5.64e-01 4.30e-01 -4.80e-04 2.46e-02 1.50e-03
[uo_nn_solve]    58 1.01e+00 9.35e-01 -3.44e-04 2.44e-02 1.19e-03
[uo_nn_solve]    59                              2.42e-02 5.96e-04
[uo_nn_solve]    k      alm       al      g'*d        f     ||g||
[uo_nn_solve]    Wall time = 1.0e-01 s.
[uo_nn_solve]    wo=[
[uo_nn_solve]         -1.4e-01,+1.4e+00,-9.5e+00,-8.8e+00,-9.1e+00
[uo_nn_solve]         +3.2e+00,-1.1e+01,+9.1e-01,+1.1e+00,+3.2e+00
[uo_nn_solve]         +1.0e+01,+1.1e-01,-1.1e+01,-6.2e+00,+4.3e+00
[uo_nn_solve]         -1.3e+01,+1.6e+01,+8.6e+00,+5.1e-01,-1.7e+01
[uo_nn_solve]         +2.4e+01,-5.0e+00,-1.4e+01,-1.3e-01,+6.1e-01
[uo_nn_solve]         +1.3e+01,-4.3e+00,-1.1e-01,+2.9e-01,-6.0e+00
[uo_nn_solve]         -7.0e+00,-8.1e+00,-6.3e+00,-8.4e+00,-3.4e+00
[uo_nn_solve]       ]
[uo_nn_solve] Accuracy.
[uo_nn_solve]    tr_accuracy = 97.6
[uo_nn_solve]    te_accuracy = 98.4
```

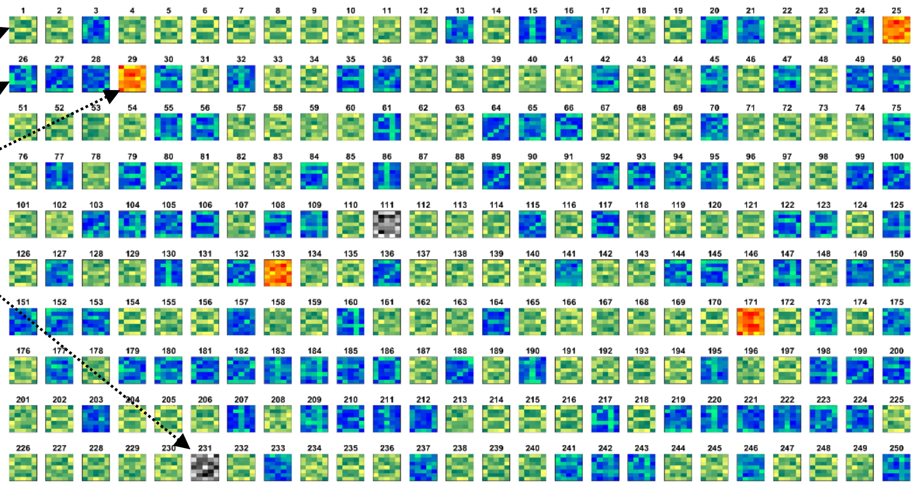
Rigth positive

Rigth negative

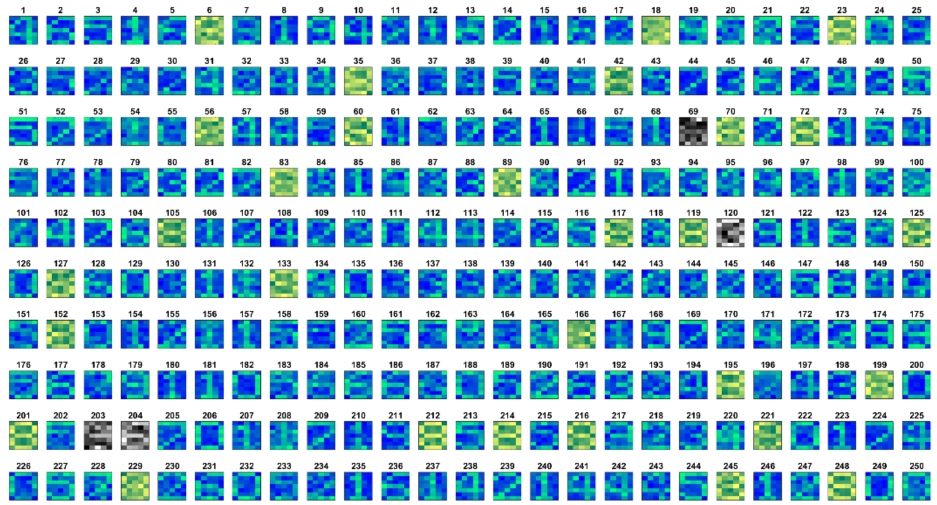
False negative

False positive

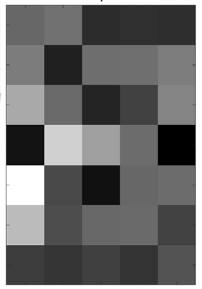
```
>> uo_nn_Xyplot(Xtr,ytr,wo)
```



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



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



Computational experiments (1/3)

- The questions to be answered in this project are:
  - i. How the three algorithms (GM/QNM/SGM) compares in terms of:
    - a) The global and local convergence when solving $\min_{w \in \mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda)$
    - b) The recognition accuracy Accuracy^{TE}.
  - ii. How the regularization parameter $\boldsymbol{\lambda}$ affects convergence and Accuracy^{TE}.
- To answer this questions a batch of instances of the SLNN problem will be solved:
  - For every **digit, 0 to 9**.
  - For every value of the regularization parameter $\boldsymbol{\lambda} \in \{\mathbf{0.0}, \mathbf{0.05}, \mathbf{0.1}\}$.
  - 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**.
- Script **`uo_nn_batch_st.m`** is provided to organize this computational experiments (see the source code included in [`uonn.zip`]).

Computational experiments (2/3)

- `uo_nn_batch.m` creates:
  - The **log file** `uo_nn_batch.log`
  - The `.csv` **file** `uo_nn_batch.csv`:

```
num_target;      la; isd;  niter;     tex; tr_acc; te_acc;        L*;
         1;  0.0000;   1;     19;  1.3016;  100.0;  100.0;  6.06e-04;
         1;  0.0000;   3;     19;  1.5917;  100.0;  100.0;  3.58e-04;
         1;  0.0000;   7;  10001;  0.5547;  100.0;  100.0;  5.55e-04;
         1;  0.0500;   1;     13;  0.9143;   99.9;  100.0;  6.69e-02;
         1;  0.0500;   3;     17;  1.5853;   99.9;  100.0;  6.69e-02;
         1;  0.0500;   7;   3001;  0.1682;   99.9;  100.0;  6.74e-02;
         1;  0.1000;   1;     12;  0.8825;   99.9;  100.0;  9.44e-02;
         1;  0.1000;   3;     12;  1.0520;   99.9;  100.0;  9.44e-02;
         1;  0.1000;   7;   3001;  0.1657;   99.9;  100.0;  9.55e-02;
         .........................................................
         0;  0.0000;   1;     79;  6.3947;   99.3;   99.1;  7.22e-03;
         0;  0.0000;   3;     39;  3.3751;   99.2;   99.1;  6.32e-03;
         0;  0.0000;   7;   8001;  0.4067;   99.3;   99.0;  6.38e-03;
         0;  0.0500;   1;     26;  1.9768;   99.1;   99.2;  1.13e-01;
         0;  0.0500;   3;     13;  1.0866;   99.1;   99.2;  1.13e-01;
         0;  0.0500;   7;   3001;  0.1674;   99.0;   99.4;  1.14e-01;
         0;  0.1000;   1;     15;  1.0707;   99.0;   99.0;  1.45e-01;
         0;  0.1000;   3;     17;  1.5931;   99.0;   99.0;  1.45e-01;
         0;  0.1000;   7;   7001;  0.3659;   99.1;   99.1;  1.45e-01;
```

Computational experiments (3/3)

- Function `uo_nn_solve` called inside **`uo_nn_batch.m`** must solve the instance corresponding to a particular combination of algorithm, digit and λ. **This is the function to be coded in this assignment**.
- The tasks to be performed by `uo_nn_solve` are:
  i. To generate the training dataset (X^{TR}, y^{TR}) and the test dataset (X^{TE}, y^{TE}) with the executable function **`uo_nn_dataset.p`**.
  ii. To find the value of w^* minimizing the loss function $\tilde{L}(w; X^{TR}, y^{TR}, \lambda)$ **with your own implementation of function uosol for the GM, QNM and SGM**.
  iii. To calculate $\mathrm{Accuracy}^{TR}$ and $\mathrm{Accuracy}^{TE}$.
- See template `uo_nn_solve_st.m.`

Report (1/2)

1. **Global convergence**: first, we want to analyse the computational convergence of every algorithm. :
   a) Define clearly, for every algorithm, how do you check global convergence. Then describe the observed global convergence for every algorithm and compare them.
   b) Is λ affecting global convergence? If so, describe the observed dependency and give an explanation.
2. **Local convergence**: now we want to see how fast these algorithm converge to the optimal solution of problem $\min_{w\in\mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda)$:
   a) Compare the speed of convergence of the three algorithms in terms of the *execution time* and *number of iterations*. Analyse the running time per iteration (tex/niter) and try to find an explanation for the different values among the three algorithms.
   b) Is λ affecting the speed of convergence? If so, describe the observed dependency and give an explanation.
3. **Accuracy**: finally, we will study the performance of every algorithm w.r.t. the recognition accuracy Accuracy^{TE} :
   a) Describe how the value of Accuracy^{TE} varies across the different combinations algorithm-λ, and how ot depends on the digit being identified.
   b) Based on the previous analysis, what do you think is the best overall combination algorithm-λ?

Report (2/2)

- 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 the following two files:
  - **`surname-student-1_surname-student-2.pdf`**: 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`**
  - **`surname-student-1_surname-student-2.zip`**: all the source 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 marking of the exercise shall take into account the formal quality, understandability and clarity, as long as the evidences provided to support the results of the analysis performed in the project (numerical values, tables, plots, …).

Grading scale

1. Global convergence.	**Points**
a) Study.	2.0
b) Relation with λ.	1.0
2. Local convergence.	
a) Study.	2.0
b) Relation with λ.	1.0
3. Accuracy.	
a) Study.	2.0
b) Conclusions	1.0
Overall quality of the report	1.0