Purpose:

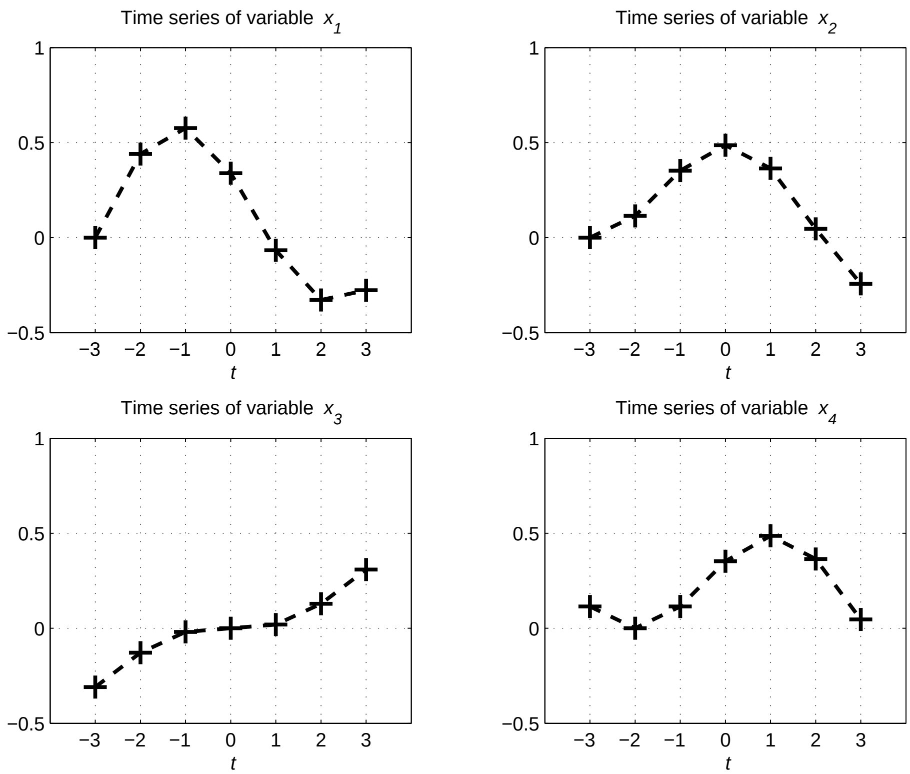
How to train an MLP neural network in MATLAB environment!

that is

For good computations,
we need good formulae
for good algorithms;
and good visualization
for good illustration
and proper testing
of good methods
and succesfull applications!

Basic Applications for MLP (I):

- time-series prediction



Derivation of learning data:

$$\begin{cases} \mathbf{x}_1 = [\, x1(-1)\;\; x2(-1)\;\; x3(-1)\;\; x4(-1)\;\; x4(-2)\,] \\ \mathbf{x}_2 = [\, x1(-2)\;\; x2(-2)\;\; x3(-2)\;\; x4(-2)\;\; x4(-3)\,] \\ \quad\vdots \end{cases} , \qquad \begin{cases} \mathbf{y}_1 = [\, x4(0)\,] \\ \mathbf{y}_2 = [\, x4(-1)\,] \\ \quad\vdots \end{cases}$$

i.e., in general:

$$\begin{cases} \mathbf{x}_t = [\, x1(-t)\;\; x2(-t)\;\; x3(-t)\;\; x4(-t)\;\; x4(-(t+1))\,] \\ \mathbf{y}_t = [\, x4(-t+1)\,] \end{cases}$$

<u>In Theory:</u> According to *Takens' theorem* (F. TAKENS, *Detecting strange attractors in turbulence*, Dynamical Systems and Turbulence, 898: 336–381, 1981) there exists, for deterministic (= funtional) systems, a diffeomorphism between a sufficiently large time window and underlying state of the dynamical system:

$$x(t) = g\left(x(t-1), x(t-2), \ldots, x(t-T)\right).$$

Basic Applications for MLP (II):

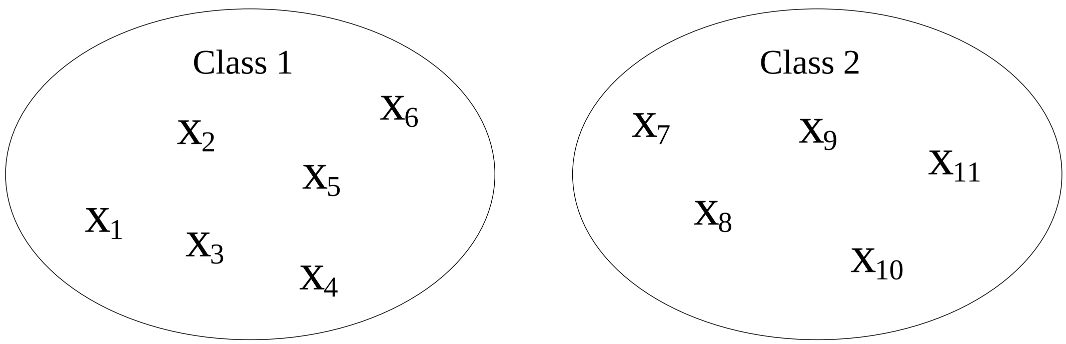
- time-series prediction (cont.)

  introduction of higher-order computing (*sigma-pi*) units:

$$\mathbf{x}_t = \left[x1(-t) \;\; x1(-t)^2 \;\; x2(-t) \;\; x2(-t)^2 \;\; x1(-t) * x2(-t) \;\; \ldots\right] \quad t = 1, 2, \ldots$$

  - basic problem: size of learning problem increased significantly!

- classification



Output coding: for kth class

$$\mathbf{y}_i^T = \mathbf{e}_k^T = [0 \;\ldots\; \overset{k's}{1} \;\ldots\; 0]^T$$

i.e., for the above example

$$\mathbf{y}_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad i = 1, \ldots, 6; \qquad \mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad i = 7, \ldots, 11.$$

Actual classifier after training the MLP:

$$\mathbf{x} \in c_k, \quad \text{where } k = \arg\max_j \; \mathbf{o}_j,$$

i.e., for the above example $\mathbf{x} \in c_1$ if $y_1 > y_2$.
More cautious approach: for suitable $0 < \theta < 1$

$$\begin{array}{ll} \text{'Class 1'}, & \text{if } o_1 > o_2 + \theta, \\ \text{'Class 2'}, & \text{if } o_2 > o_1 + \theta, \\ \text{'Unclassified'}, & \text{otherwise.} \end{array}$$

Basic Applications for MLP (III):

- data compression (i.e., nonlinear PCA)
  - We try to construct *coder* $\mathcal{K}$ and *decoder* $\mathcal{D}$ according to basic principle

    $$\mathbf{x}_i \sim \mathcal{D}(\mathcal{K}(\mathbf{x}_i))$$

    for all data vectors $\mathbf{x}_i, \ i = 1, \ldots, N$.
  - Multilayered Perceptron approach:

    $$\mathbf{x}_i \sim \mathbf{W}^4\, \widehat{\mathbf{F}}^2(\mathbf{W}^3\, \widehat{\mathbf{W}}^2\, \widehat{\mathbf{F}}^1(\mathbf{W}^1\, \hat{\mathbf{x}})),$$

    where `size(W2,1) << size(X,2)`.
  - yields very large learning problem
  - quality of coding&decoding can be evaluated from the residual of cost function $\longrightarrow$ a way to control the choice of `size(W2,1)`
- other applications (! = exists, ? = worth testing)
  - autopilots for moving vehicles(!)
  - functional algorithms(?): *sort, schedule* etc.
  - automatic differentiation of an optimization problem(!)
  - data inputation(?!)
  - $\ldots$(?)

Improving the performance of MLP:

Preprocessing: rescaling, PCA, normalization,...

Restricting the generality:

basic principle: try to keep the unknown weights in the neighborhood of zero, i.e. around the linear region of the activation fuctions

realization: penalization of weight values

$$\mathcal{J}(\{\mathbf{W}^l\}) = \frac{1}{2N}\sum_{i=1}^{N} \left\|\mathcal{N}(\{\mathbf{W}^l\})(\mathbf{x}_i) - \mathbf{y}_i\right\|^2 + \frac{\beta}{2}\sum_{l=1}^{L}\sum_{(i,j)\in I_l} |\mathbf{W}^l_{i,j}|^2,$$

where I_l is defined as

$$I_l = \begin{cases} \{(i,j) \,:\, 1 \le i \le n_l,\; 0 \le j \le n_{l-1}\}, & l < L, \\ \{(i,j) \,:\, 1 \le i \le n_l,\; 1 \le j \le n_{l-1}\}, & l = L. \end{cases}$$

- easily added into sensitivity analysis of learning problem
- choise of new free (hyper)parameter $\beta \ge 0$ creates new problem, but already a decent value can be helpful

Combination of MLP and RBFN:

- RBFN (*Radial-Basis Function Network*) approximates (uniformly) input-output mapping using local basis functions, most commonly *Gaussian kernels*:

$$\mathbf{o} = \mathbf{W}\,[\exp(-\frac{\|\mathbf{x}-\mu_1\|^2}{\delta_1^2}) \ldots \exp(-\frac{\|\mathbf{x}-\mu_i\|^2}{\delta_i^2}) \ldots \exp(-\frac{\|\mathbf{x}-\mu_m\|^2}{\delta_m^2})]^T$$

  assuming that centroids $\{\mu_i\}_{i=1}^m$ and standard deviations $\{\delta_i\}_{i=1}^m$ are given.

- SQUARE-MLP (*square unit augmented, radially extended, multilayer perceptron,* FLAKE) tries to combines good properties of MLP (global approximation) and RBFN (local representation) simply by (cf. *sigma-pi* units):

$$\begin{aligned} \mathbf{x}_i &= \left[(\mathbf{x}_i)_1 \;\; (\mathbf{x}_i)_1^2 \;\; \ldots \;\; (\mathbf{x}_i)_k \;\; (\mathbf{x}_i)_k^2 \;\; \ldots \;\; (\mathbf{x}_i)_{n_0} \;\; (\mathbf{x}_i)_{n_0}^2\right] \quad i = 1,\ldots,N, \\ &= \left[(\mathbf{x}_i)_1 \;\; \ldots \;\; (\mathbf{x}_i)_k \;\; \ldots \;\; (\mathbf{x}_i)_{n_0} \;\; (\mathbf{x}_i)_1^2 \;\; \ldots \;\; (\mathbf{x}_i)_k^2 \;\; \ldots \;\; (\mathbf{x}_i)_{n_0}^2\right]. \end{aligned}$$