Artificial Intelligence

Neural Networks

Lesson 12: Learning Vector Quantization

Vincenzo Piuri

Università degli Studi di Milano

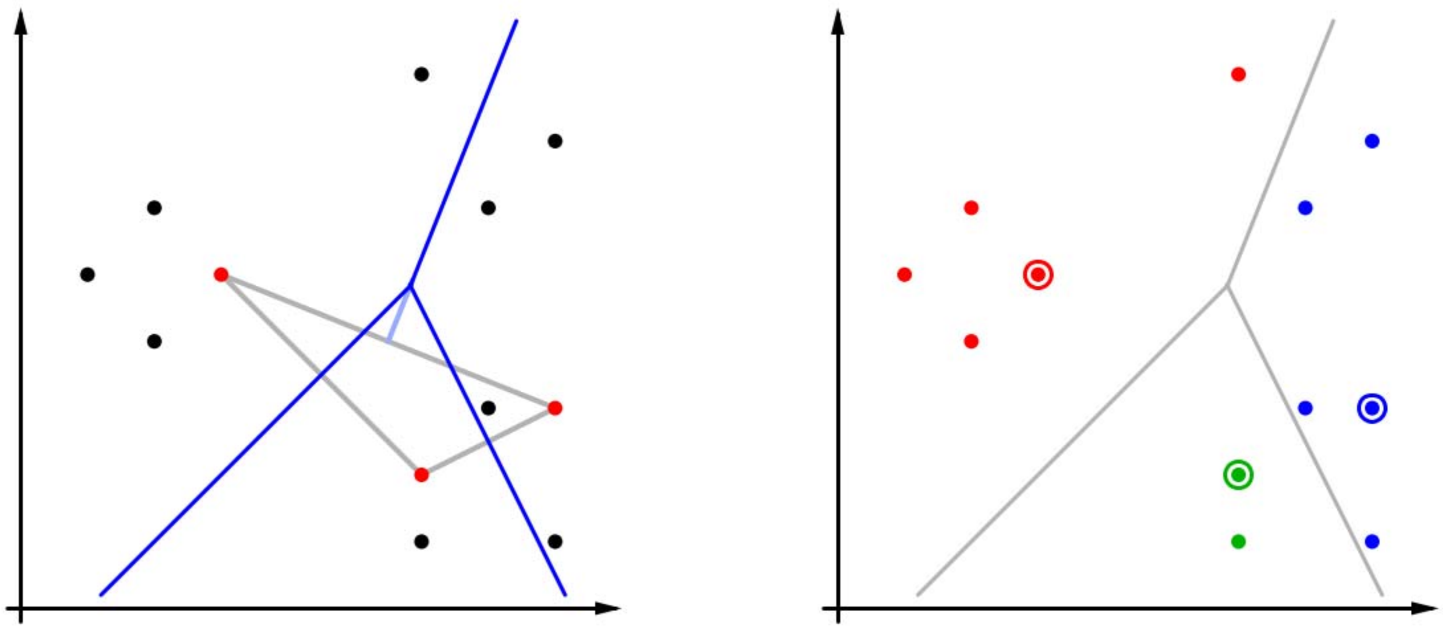
Contents

Learning Vector Quantization (1)

- So far: fixed learning tasks
  - The data consists of input/output pairs
  - The objective is to produce desired output for given input
  - This allows to describe training as error minimization
- Now: **free learning tasks**
  - The data consists only of input values/vectors
  - The objective is to produce similar output for similar input (clustering)

Learning Vector Quantization (2)

- **Delaunay Triangulation**: simple triangle (shown in gray on the left)
- **Voronoi Diagram**: mid-perpendiculars of the triangle's edges (shown in blue on the left, in gray on the right)

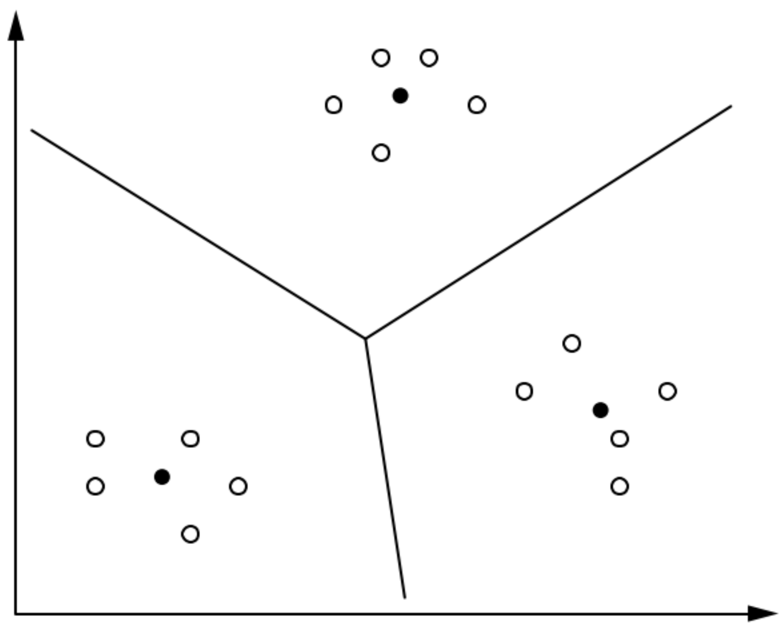


Learning Vector Quantization (3)

- **Learning Vector Quantization**
  - Find a suitable quantization (many-to-few mapping, often to a finite set) of the input space, e.g. a tessellation of a Euclidean space
  - Training adapts the coordinates of so-called reference or codebook vectors, each of which defines a region in the input space

Learning Vector Quantization (4)

- Finding clusters in a given set of data points
  - Data points are represented by empty circles (◦)
  - Cluster centers are represented by full circles (•)

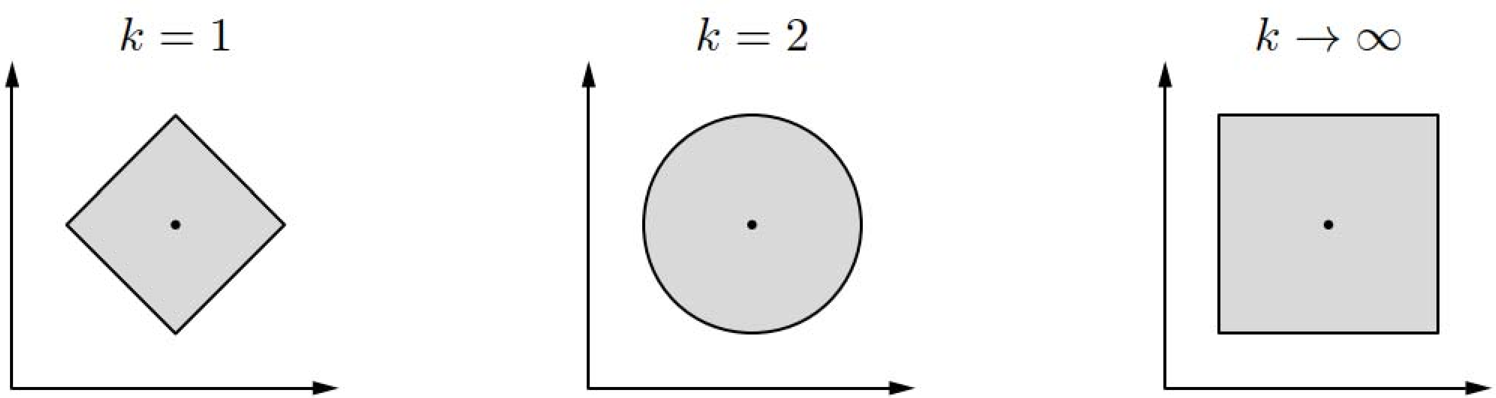


Learning Vector Quantization Nets (1)

- A **Learning Vector Quantization Network (LVQ)** is a feed-forward 2-layered neural network
- It can be viewed as a RBF network with hidden layer used as output layer
- The network input function of each output neuron is a distance function of the input vector and the weight vector
  - $\forall u \in U_{out}: f_{net}^{(u)}\left(\overrightarrow{w_u}, \overrightarrow{in_u}\right) = d(\overrightarrow{w_u}, \overrightarrow{in_u})$
  - $d(\vec{x}, \vec{y}) = 0 \iff \vec{x} = \vec{y}$
  - $d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x})$ (Symmetry)
  - $d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z})$ (Triangle inequality)

Learning Vector Quantization Nets (2)

- Distance functions: **Minkowski Family**
  - $d_k(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^k\right)^{\frac{1}{k}}$
  - $k = 1$: Manhattan or city block distance
  - $k = 2$: Euclidean distance
  - …
  - $k \to \infty$: Maximum distance



Learning Vector Quantization Nets (3)

- The activation function of each output neuron is a **radial function**
  - Monotonically decreasing function
    - $f: \mathbb{R}_0^+ \rightarrow [0,1]$ with $f(0) = 1$ and $\lim_{x \to \infty} f(x) = 0$

Learning Vector Quantization Nets (4)

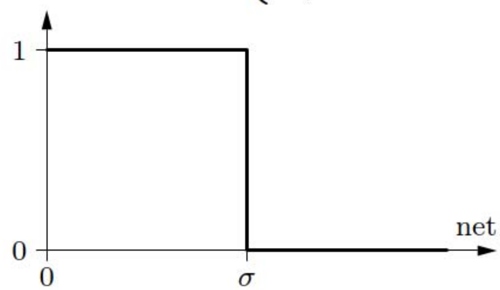
- The output function of each output neuron is not a simple function of the activation of the neuron
  - It considers the activations of all output neurons
  - $f_{out}^{(u)}(act_u) = \begin{cases} 1 & \text{if } act_u = \max\limits_{v \in U_{out}} act_v \\ 0 & \text{otherwise} \end{cases}$
  - If more than one unit has the maximal activation, one is selected at random to have an output of 1, all others are set to output 0: **winner-takes-all principle**

Learning Vector Quantization Nets (5)

- Radial activation functions

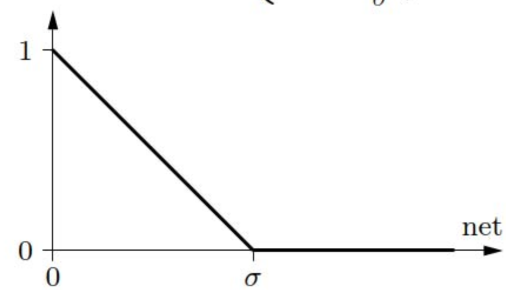
rectangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1, & \text{otherwise.} \end{cases}$$



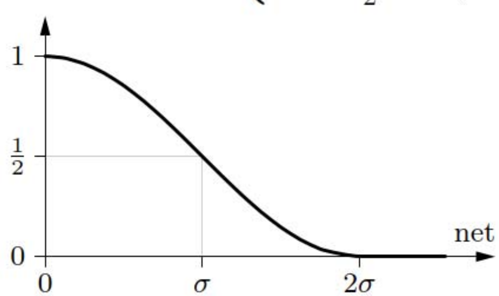
triangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1 - \frac{\text{net}}{\sigma}, & \text{otherwise.} \end{cases}$$



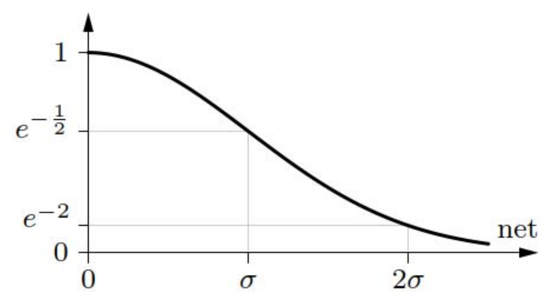
cosine until zero:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > 2\sigma, \\ \frac{\cos\left(\frac{\pi}{2\sigma}\text{net}\right)+1}{2}, & \text{otherwise.} \end{cases}$$



Gaussian function:

$$f_{\text{act}}(\text{net}, \sigma) = e^{-\frac{\text{net}^2}{2\sigma^2}}$$



Learning rules (1)

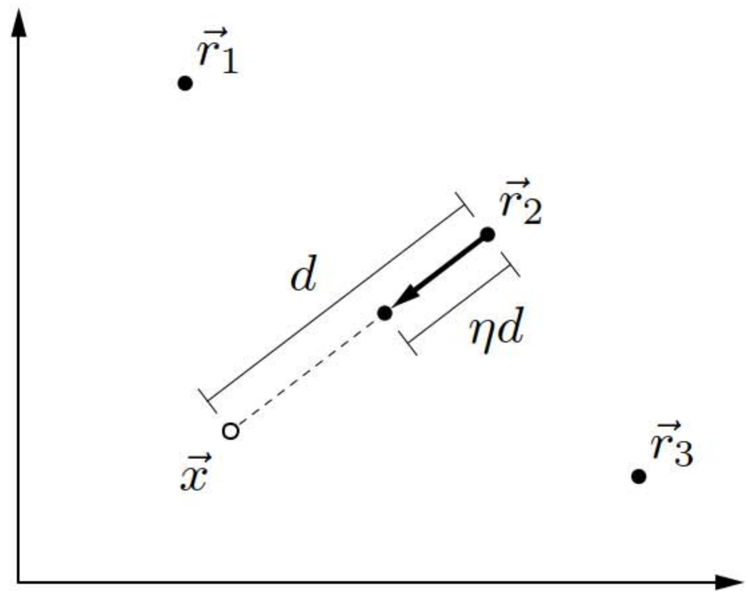
- Adaptation of reference vectors
  - For each training pattern find the closest reference vector
  - Adapt only this reference vector (winner neuron)
  - For classified data the class may be considered
  - Each reference vector is assigned to a class

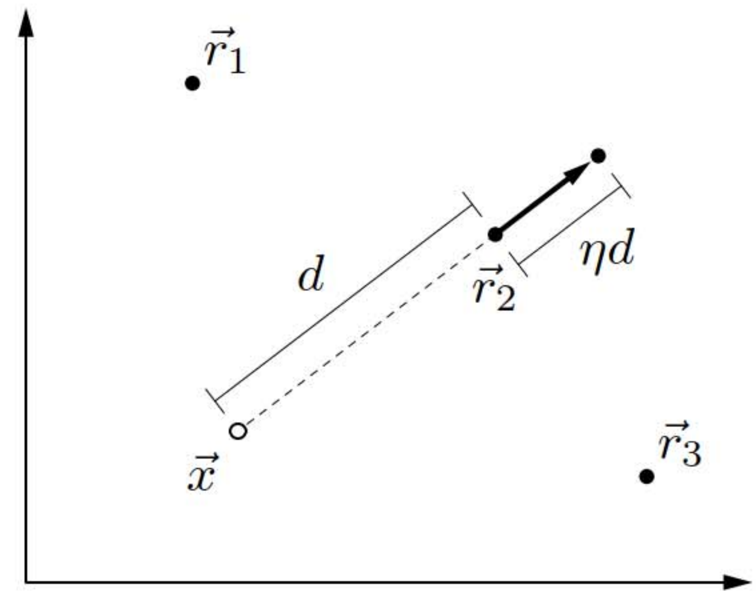
Learning rules (2)

- **Attraction rule** (data point and reference vector have same class)
  - $\vec{r}^{(new)} = \vec{r}^{(old)} + \eta(\vec{x} - \vec{r}^{(old)})$

- **Repulsion rule** (data point and reference vector have different class)
  - $\vec{r}^{(new)} = \vec{r}^{(old)} - \eta(\vec{x} - \vec{r}^{(old)})$

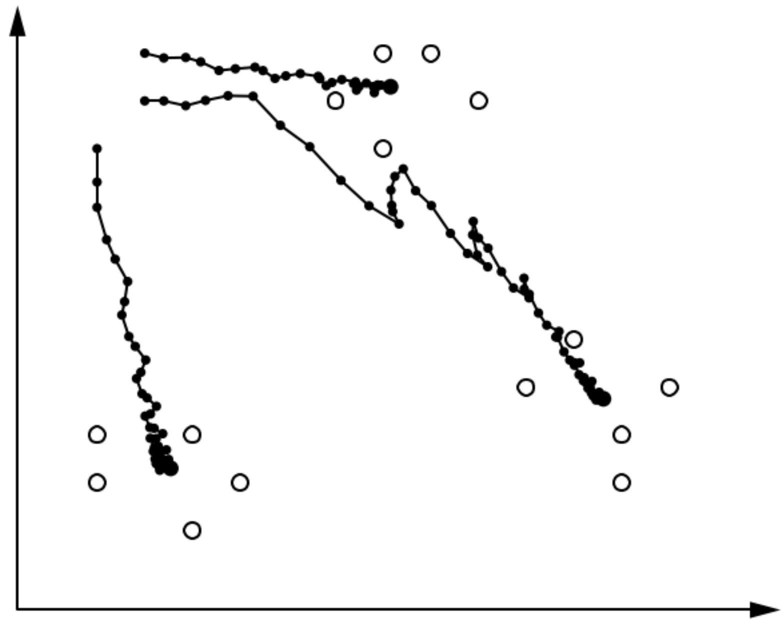
Learning rules (3)



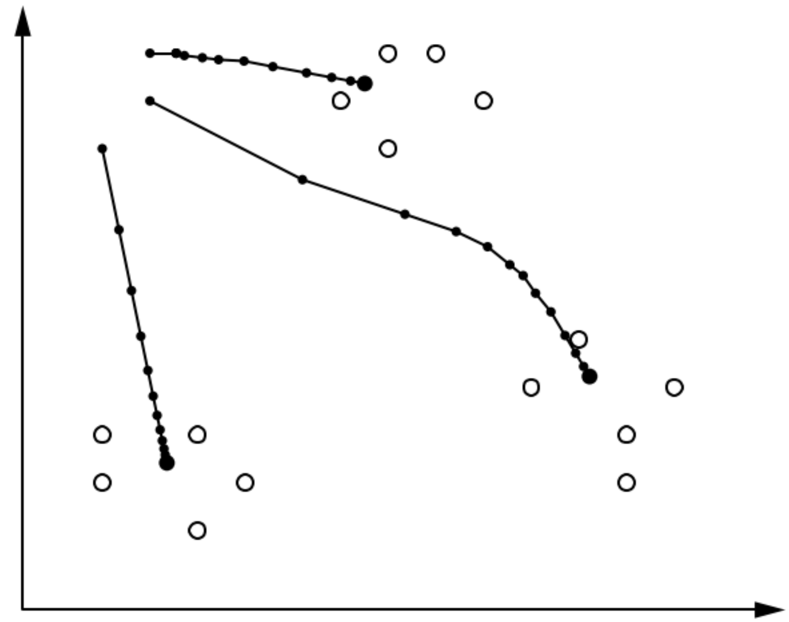


- Adaptation of reference vectors

Learning rules (4)



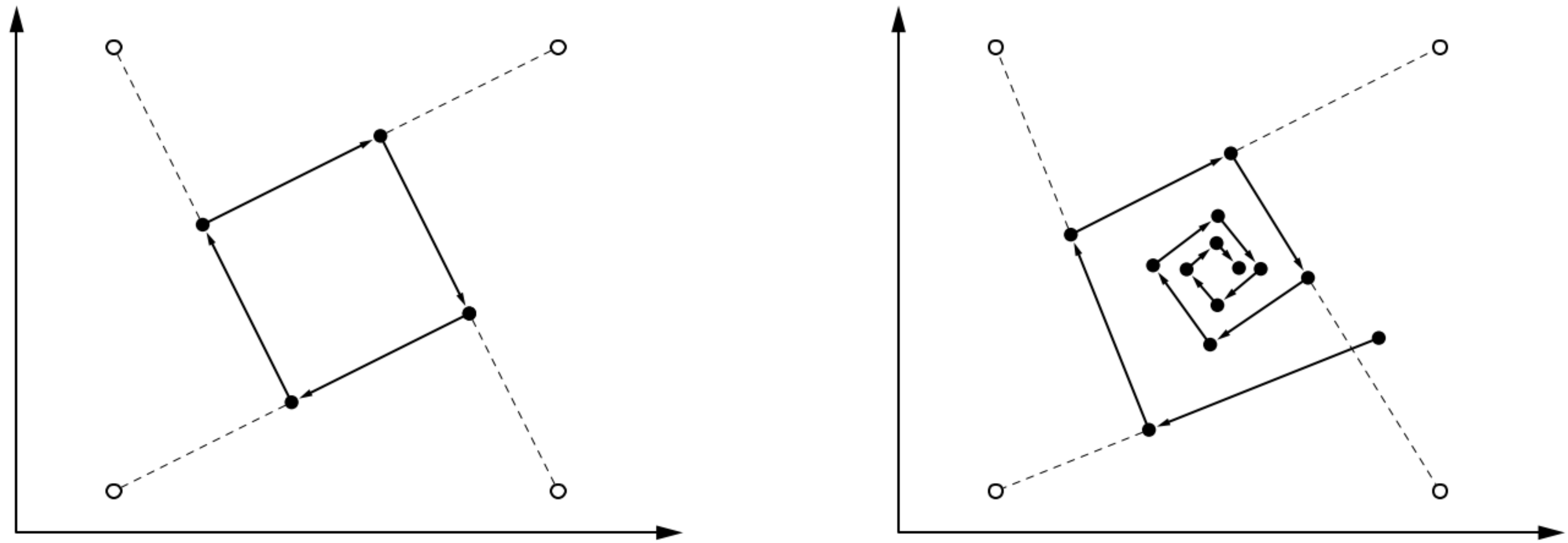
Online training



Batch training

- Adaptation of reference vectors

Learning rules (5)



- Fixed learning rate can lead to oscillations
  - Solution: **time dependent learning rate**

Learning rules (6)

- Update rule for **classified data**
  - Update not only the one reference vector that is closest to the data point (the winner neuron), but update the two closest reference vectors
  - All other reference vectors remain unchanged

Learning rules (7)

- Standard learning vector quantization may drive the reference vectors further and further apart
- **Window rule**
  - Update only if the data point $\vec{x}$ is close to the classification boundary

$$\min\left(\frac{d(\vec{x},\vec{r}_j)}{d(\vec{x},\vec{r}_k)},\frac{d(\vec{x},\vec{r}_k)}{d(\vec{x},\vec{r}_j)}\right) > \theta, \qquad \text{where} \qquad \theta = \frac{1-\xi}{1+\xi}.$$

  - ξ is a parameter that has to be specified by a user
  - ξ describes the “width” of the window around the classification boundary
  - The update ceases once the classification boundary has been moved far enough away

Soft Learning Vector Quantization

- Use soft assignments instead of winner-takes-all
  - Assumption: given data was sampled from a mixture of normal distributions
- Closely related to clustering by estimating a **mixture of Gaussians**
  - (Crisp or hard) learning vector quantization is an "online version" of C-means clustering
  - Soft learning vector quantization is an "online version" of estimating a mixture of Gaussians

Expectation Maximization (1)

- Mixture of Gaussians
  - Assumption: Data was generated by sampling a set of normal distributions
  - We assume that the probability density can be described as

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

$\mathbf{C}$	is the set of cluster parameters
$\vec{X}$	is a random vector that has the data space as its domain
Y	is a random variable that has the cluster indices as possible values (i.e., $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$ and $\mathrm{dom}(Y) = \{1, \ldots, c\}$)
$p_Y(y; \mathbf{C})$	is the probability that a data point belongs to (is generated by) the y-th component of the mixture
$f_{\vec{X}\|Y}(\vec{x}\|y; \mathbf{C})$	is the conditional probability density function of a data point given the cluster (specified by the cluster index y)

Expectation Maximization (2)

- Maximum likelihood estimation of the cluster parameters
  - The likelihood function is difficult to optimize

$$L(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}) = \prod_{j=1}^{n} \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j | y; \mathbf{C}),$$

  - Approach: Assume "hidden" variables Y_j stating the clusters that generated the data points $\vec{x}_j$

$$L(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) = \prod_{j=1}^{n} p_{Y_j}(y_j; \mathbf{C}) \cdot f_{\vec{X}_j|Y_j}(\vec{x}_j | y_j; \mathbf{C}).$$

  - Problem: We do not know the values of Y_j

Expectation Maximization (3)

- Approach
  - See the Y_j as random variables (the values y_j are not fixed) and consider a probability distribution over the possible values
  - $L(\mathbf{X}, \vec{y}; \mathbf{C})$ becomes a random variable, even for a fixed data set $\mathbf{X}$ and fixed cluster parameters $\mathbf{C}$
  - Try to maximize the expected value of $L(\mathbf{X}, \vec{y}; \mathbf{C})$ or $\ln L(\mathbf{X}, \vec{y}; \mathbf{C})$ (expectation maximization)

Expectation Maximization (4)

- Find the cluster parameters

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmax}}\, E([\ln] L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}),$$

  – Maximize the expected likelihood

$$E(L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C})$$

  – Or maximize the expected log-likelihood

$$E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}).$$

Expectation Maximization (5)

- Still difficult to optimize directly
  - Solution: use the equation as an iterative scheme, fixing C in some terms
  - Iteratively compute better approximations

Expectation Maximization (6)

- Iterative scheme for expectation maximization
  - Choose some initial set C_0 of cluster parameters and then compute

$$
\begin{aligned}
\mathbf{C}_{k+1} &= \underset{\mathbf{C}}{\operatorname{argmax}}\, E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}_k) \\
&= \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}_k) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}).
\end{aligned}
$$

  - Each EM iteration increases the likelihood of the data and the algorithm converges to a local maximum of the likelihood function

Expectation Maximization (7)

- Core Iteration Formula

$$\mathbf{C}_{k+1} = \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C})$$

1. **Expectation step**
   – For all data points $\vec{x_j}$: Compute for each normal distribution the probability $p_{Y_j|\vec{X_j}}(i|\vec{x_j}; \mathbf{C}_k)$ that the data point was generated from it
2. **Maximization step**
   – For all normal distributions: estimate the parameters by standard maximum likelihood estimation using the probabilities ("weights") assigned to the data points w.r.t. the distribution in the expectation step