



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Unsupervised Learning K-Means Clustering Principal Component Analysis (PCA) Autoencoders

Prof. Vasilis Maglaris <u>maglaris@netmode.ntua.gr</u> <u>www.netmode.ntua.gr</u> Room 002, New ECE Building Tuesday February 25, 2025

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Some Special Neural Network Configurations - Supervised Learning

**Convolutional Neural Networks (CNN)**: *Multilayer Perceptron* category preferred for classification of two-dimensional examples (e.g. *pattern recognition* of images) via *supervised learning*. The CNN simplification results by decoupling the net into loosely connected parts with common *receptive fields* among subsets of input element features and exhibiting *convolutional induced local fields* 

**Support Vector Machines (SVM)**: Binary classification into regions of maximum linear separation via *supervised learning*. Regions are separated by linear *neutral border zones* as wide as possible, defined by *support vectors* as shown in the two-dimensional adjacent figure. Sample elements belong in two classes, depicted as blue squares and circles. In case of *non-separable patterns*, regions result from the training sample that minimize classification errors



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Some Special Neural Network Configurations – Unsupervised Learning & LLM *K*- Means Clustering: Self-organization of observed sample elements $\mathbf{x}_i$ into *K* clusters via unsupervised learning, e.g. clustering based on proximity of Euclidean Distances $\|\mathbf{x}_i - \mathbf{x}_i\|^2$

**Principal Components Analysis (PCA)**: Unsupervised learning used to map sample vectors of high dimensionality, e.g. images,  $\mathbf{x} = [x_1 \ x_2 \ ... \ x_m]^T$  into output vectors  $\mathbf{y} = [y_1 \ y_2 \ ... \ y_l]^T$  of  $l \ll m$  Principal Components by selecting the most important features (*feature engineering*). Used to overcome the curse of dimensionality for image reconstruction, pattern classification...

**Self-Organizing Maps (SOM)**: Neurons are placed on vertices of a two-dimensional *lattice* and converge into maps of location-based significant traits (features) via *competitive unsupervised learning*. Identification of *active neurons* is learned by boosting or attenuating weights of paths between neurons in the lattice, aiming at the final winner selection (*winner takes all*)

Large Language Models (LLM): Used to identify missing words (*masked tokens*) or sentences, generate (via *GenAI*) answers to *chatbot* and/or *search engine* queries, *translate* to alternate natural languages... They employ *Natural Language Processing* (*NLP*) algorithms (e.g. *Attention Mechanism* based *Transformers*), may rely on special hardware (e.g. *GPUs*) and can require extensive *pre-training* in massive datacenters (possibly months of parameter tuning with billions of data elements). Offered (free or for-a-fee) to tens of millions of end-users as a *cloud service* via the Web, usually with a *reasoning* option. Users may upload reduced models in their machines (e.g. laptops). Very recent killer applications: *ChatGPT* (OpenAI), *DeepSeek*. Risks include prediction errors - hallucinations, IPR infringement, plagiarism, excessive reliance to black-box *Artificial Intelligence* methods...

Tutorial by Mirella Lapata <u>https://www.youtube.com/watch?v=\_6R7Ym6Vy\_I&t=1894s</u>

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Unsupervised Learning

- Unsupervised learning is based on estimating *a-priori* probabilities *p*(**x**) of sample elements (vectors) **x** = [*x*<sub>1</sub> *x*<sub>2</sub>... *x<sub>m</sub>*]<sup>T</sup> with *m* features *x<sub>i</sub>* (e.g. *K*-Means Clustering – selection of *K* centers of gravity or cluster centroids and assignment of vectors **x** into closest centroids)
- It is based on input *feature* statistics estimated from *unlabeled training examples* and assumptions of the environment behavior (e.g. *Hebb's* rule). The system assigns an output  $y = h_w(x)$  (e.g. compressed image or class of x) consistent with models inferred from user requirements and conforming to pre-stored experience
- Apart from training & validation examples used to design the system, test data are normally added to assess a trained model  $h_w(\cdot)$  generalization capability and overfitting risk
- Unsupervised learning is a widely employed method of self organization (e.g. *Self-Organizing Maps SOM, Autoencoders*) and of *principal component filtering* for efficient storage processing classification of *sample vectors* with massive number of features (typical in *speech text image processing* applications and *pattern recognition* models **Note:** Definition of Sample, Sample Elements & Sample Space in Statistics A sample is defined as a subset of a superset, referred to as *sample space*, that approximately exhibits its statistical properties. It consists of *N* sample elements (examples), typically vectors  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$  with coordinates  $x_i$  encoding the *m* features (characteristics) of  $\mathbf{x}$

**K**- Means Clustering via Unsupervised Learning

**Encoding** into *K* clusters of *N* unlabeled training examples  $\mathbf{x}(n) = [x_1(n) \ x_2(n) \dots \ x_m(n)]^T$ 

- Determination of Encoder C(n) = j:  $\mathbf{x}(n)$ , n = 1, 2, ..., N belongs to cluster j = 1, 2, ..., K
- Symmetric Measure of Similarity:  $d(\mathbf{x}(n), \mathbf{x}(n')) = d(\mathbf{x}(n'), \mathbf{x}(n))$ Example: Euclidean Distance,  $d(\mathbf{x}(n), \mathbf{x}(n')) \triangleq ||\mathbf{x}(n) - \mathbf{x}(n')||^2$
- Estimation of centroid μ̂<sub>j</sub> as center of gravity of cluster j = 1,2,...,K: Mean Euclidean Distance of x (i) from μ̂<sub>j</sub> for all encoder options C (n) = j
- Cost:  $J(C) = \frac{1}{2} \sum_{j=1}^{K} \sum_{C(n)=j} \sum_{C(n')=j} \|\mathbf{x}(n) \mathbf{x}(n')\|^2 = \sum_{j=1}^{K} \sum_{C(n)=j} \|\mathbf{x}(n) \widehat{\boldsymbol{\mu}}_j\|^2$
- Minimization Criterion: Variance  $\hat{\sigma}_{j}^{2} \triangleq \sum_{\substack{C(n)=j \ j}} \|\mathbf{x}(n) \hat{\boldsymbol{\mu}}_{j}\|^{2}$ ,  $\min J(C) = \min \sum_{j=1}^{K} \hat{\sigma}_{j}^{2}$ Self-Organization of N Training Sample Points into K Clusters
- Initialization: Arbitrary selection of hyperparameter  ${\boldsymbol K}$
- Assign the N training sample points  $\mathbf{x}(n)$  to the closest centroid
- Update centroid selection  $\hat{\mu}_j$ , j = 1, 2, ..., K & re-evaluate assignment of encoders C(n) = j
- Efficient & easy to code algorithm but with no formal convergence proof
- The choice of K may involve several **trials** and **variance** comparisons by increasing K up to knee of the minimum cost



**Principal Components Analysis - PCA** 

# The Curse of Dimensionality:

In a sample space of vectors  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$ , encoding of features (categorical or continuous) may lead to a large number of dimensions m (e.g. number of pixels in an image) Rule of Thumb: For statistically meaningful encoding of the m features, it is empirically required a number of N sample vectors that is a multiple of m (e.g.  $N \gg 5m$ , <u>https://en.wikipedia.org/wiki/Curse\_of\_dimensionality</u>)

# **Reduction of Dimensionality - Principal Components:**

A vector space of  $\mathbf{x} = [x_1 \ x_2 \ ... \ x_m]^T$  the *m* coordinates can be mapped into orthogonal (uncorrelated) *Principal Components*, ordered by their significance. The next step is to truncate insignificant components and obtain an output vector  $\mathbf{y} = [y_1 \ y_2 \ ... \ y_l]^T$  by selecting the  $l \ll m$  principal features

### **Methods for Selecting Principal Components:**

- The <u>Covariance Method</u>: Statistical analysis of the training sample space, linear transformation of its *m* coordinates using an *orthonormal* basis and selecting the *l « m* principals via <u>Linear Algebra</u> methods (similar to the <u>Karhunen Loève Expansion</u> with orthogonal deterministic basis functions in *Time-Series Theory* <u>https://en.wikipedia.org/wiki/Kosambi%E2%80%93Karhunen%E2%80%93Lo%C3%A8ve\_t heorem</u>)
- The <u>Hebbian Learning Method</u>: Via self-organized Neural Network models with Hebbian local tuning in unsupervised learning

**Orthonormal Transformation to Principal Components (1/3)** 

### **Covariance Method - Definitions**

- Input: Sample elements (vectors)  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$  of *m* features encoded in  $x_i$  values
- **Coordinates**  $x_i$ : Sample Values of Random Variables  $X_i$ , the coordinates of random vectors  $\mathbf{X} = [X_1 \ X_2 \dots \ X_m]^T$  of the training sample space. We assume that  $\mathbf{E}[\mathbf{X}] = \mathbf{E}[X_i] = 0$
- **Correlation Matrix:** The symmetric matrix  $(m \times m) \mathbf{R} = E[\mathbf{X} \mathbf{X}^T]$  with elements  $E[x_i x_j]$ , eignevectors  $\mathbf{q}_j$  and eigenvalues  $\lambda_j$ :  $\mathbf{R}\mathbf{q}_j = \lambda_j \mathbf{q}_j$ , j = 1, 2, ..., m in decreasing order of  $\lambda_j$  $\lambda_j \mathbf{q}_j^T \mathbf{q}_k = \begin{cases} 1, \ k = j \\ 0, \ k \neq j \end{cases}$  and  $\mathbf{q}_j^T \mathbf{R} \mathbf{q}_k = \begin{cases} \lambda_j, \ k = j \\ 0, \ k \neq j \end{cases}$
- **Principal Components:** The eigenvectors  $\mathbf{q}_j$  define Orthonormal Principal directions that via linear transformation map a random vector  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$  of m coordinates  $x_i$  into the random vector  $\mathbf{a} = [a_1 \ \alpha_2 \dots \ \alpha_m]^T = [\mathbf{x}^T \mathbf{q}_1 \ \mathbf{x}^T \mathbf{q}_2 \dots \ \mathbf{x}^T \mathbf{q}_m]$  of m coordinates  $a_i$  referred to as Principal Components
  - ✓ The order of  $\alpha_j$ 's follows the decreasing order of  $\lambda_j = \mathbf{q}_j^T \mathbf{R} \mathbf{q}_j = var(A_j) \triangleq \sigma_j^2$ , with  $A_j$  a random variable with sample value  $\alpha_j$
  - ✓ The original coordinates  $x_i$  are uniquely deduced from the Principal Components:

$$\mathbf{x} = \sum_{j=1}^{m} a_j \mathbf{q}_j$$

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Orthonormal Transformation to Principal Components (2/3)

By ignoring *principal components* with smaller variances  $\sigma_j^2 = \lambda_j$ , j = l + 1, l + 2,...,m we can approximate (encode) vector  $\mathbf{x}$  with  $\hat{\mathbf{x}}$  of reduced dimensionality l < m

$$\hat{\mathbf{x}} = [\hat{x}_1 \ \hat{x}_2 \dots \ \hat{x}_l] = [\mathbf{q}_1 \ \mathbf{q}_2 \dots \mathbf{q}_l] [a_1 \ \alpha_2 \dots \ \alpha_l]^{\mathrm{T}} = \sum a_j \mathbf{q}_j \text{ for } l < m$$



- The total variance of the *m* independent random variables  $\varepsilon X_j$  is  $\sum_{j=1}^m \sigma_j^2 = \sum_{j=1}^m \lambda_j$
- The total variance of the *l* Principal Components  $A_j$  is  $\sum_{j=1}^{l} \sigma_j^2 = \sum_{j=1}^{l} \lambda_j$ 
  - ⇒ The total variance of the error  $\mathbf{e} = \mathbf{x} \hat{\mathbf{x}}$  is  $\lambda_{l+1} + \lambda_{l+2} + ... + \lambda_m$  (the principal components with the smaller variances)

Orthonormal Transformation to Principal Components (3/3)

### Application of PCA for Image Compression & Pattern Recognition of Handwritten Numbers

**Training Sample:** 

Scanned images of handwritten numbers  $\{0, 1, \dots, 9\}$ 

N = 1700 elements/number

 $m = 32 \times 32 = 1024$  pixels/image (features)

Column 1: Encoding with *m* binary digits (black or white/pixel) **Column 2:** Sample averages for normalization

Evaluation of l = 64 principal eigenvectors of the  $m \times m = (1024) \times (1024)$  correlation matrix after normalization (subtraction of sample averages)

**Reconstruction of Images with**  $l \le 64$  **Principal Components**  $l \ll m = 32 \times 32 = 1024$ Column 3: l = 1Column 4: l = 5**Column 5**: l = 16Column 6: l = 32 (acceptable identification of numbers) Column 7: l = 64 (perfect reproducibility under significant

compression,  $1024 \rightarrow 64$ )

Hebbian Learning of 1<sup>st</sup> Principal Component (1/2)

### Self-Organized Feature Analysis

**Rule of Hebb**: If signals (states) in the borders of a neural synapsis *i* are synchronously updated in step *n*, the synaptic weight  $w_i(n)$  increase. Else it tends to zero (inspired from neuropsychology learning context) **Competition Principle**: The most active synapses tend to eliminate weak ones

### Hebbian-based Maximum Eigenfilter

Linear Neural Network:

 $y(n) = v(n) = \sum_{i=1}^{m} w_i(n) x_i(n)$  at step n

### **Hebbian Learning**:

Weights increase at step  $n \rightarrow n + 1 \le N$  if  $y(n) x_i(n) > 0$ 

 $w_i(n+1) = w_i(n) + \eta y(n) x_i(n), i = 1,2,...,m, \eta$  learning-rate hyperparameter

To enforce stabilization (avoid unlimited growth) in every step (based on the **Competition Pr<u>inciple</u>**) we normalize by summing over all synapses associated with the neuron:

$$w_{i}(n+1) = \frac{w_{i}(n) + \eta y(n) x_{i}(n)}{\left(\sum_{k=1}^{m} [w_{k}(n) + \eta y(n) x_{k}(n)]^{2}\right)^{1/2}} \approx w_{i}(n) + \eta y(n) [x_{i}(n) - y(n) w_{i}(n)]$$
  
the approximation is valid for small values of  $\eta$ 

 $x_i(n)$ 

 $\eta y(n)$ 

 $w_i(n+1)$ 

### Signal-flow Graph of Maximum Eigenfilter with Normalization

Positive feedback  $y(n) x_i(n)$  is countered by  $y(n) w_i(n)$ 

$$w_i(n+1) = w_i(n) + \eta y(n) x'_i(n), \ x'_i(n)$$



y(n)

 $o w_i(n)$ 

Hebbian Learning of 1<sup>st</sup> Principal Component (2/2)

**Convergence Issues of Self-Organized Algorithm** 

**Definitions of Vectors** 

 $\mathbf{x}(n) = [x_1(n) \ x_2(n) \ \dots \ x_m(n)]^{\mathrm{T}}$  $\mathbf{w}(n) = [w_1(n) \ w_2(n) \ \dots \ w_m(n)]^{\mathrm{T}}$ 

Unsupervised Learning via Self-Organization Algorithm:

$$y(n) = \mathbf{w}^{\mathrm{T}}(n) \mathbf{x}(n), \ \mathbf{w}(n+1) = \mathbf{w}(n) + \eta y(n) [\mathbf{x}(n) - y(n) \mathbf{w}(n)] \Rightarrow$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta [\mathbf{x}(n) \mathbf{x}^{\mathrm{T}}(n) \mathbf{w}(n) - \mathbf{w}^{\mathrm{T}}(n) (\mathbf{x}(n) \mathbf{x}^{\mathrm{T}}(n)) \mathbf{w}(n) \mathbf{w}(n)]$$

- Factors x (n) x<sup>T</sup>(n) represent the Correlation Matrix R = E[X X<sup>T</sup>] at training iteration step n → n + 1 ≤ N without mean values. It leads to convergence properties of the algorithm using non-linear stochastic difference equations (beyond the scope of the lectures)
- There is no external influence to the self-organized unsupervised learning algorithm, except the a-priori setting of the training hyperparameter  $\eta$

### STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Generalized Hebbian-based Principal-Component Analysis (1/3)

Generalization of Hebbian-based Maximum Eigenfilter:  
Linear Feedforward Single Layer Neural Network of 
$$l$$
  
output neurons associated with the most important  
Principal Components of input sample vectors of  
dimensionality  $m, l < m$   
 $y_j(n) = v_j(n) = \sum_{i=1}^{m} w_{ji}(n) x_i(n), j = 1, 2, ..., l$ 

Hebbian Learning: Weights  $w_{ji}(n)$  from input  $x_i(n)$ , i = 1, 2, ..., m to the Principal Component  $y_j(n)$ , j = 1, 2, ..., l change by  $\Delta w_{ji}(n)$  in iteration  $n \to n + 1$  $\Delta w_{ji}(n) = \eta \left[ y_j(n) x_i(n) - y_j(n) \sum_{k=1}^j w_{ki}(n) y_k(n) \right]$ , j = 1, 2, ..., l & i = 1, 2, ..., m

**Generalized Hebbian-based Principal-Component Analysis (2/3)** 



**Generalized Hebbian-based Principal-Component Analysis (3/3)** 

$$\mathbf{x'}(n) = \mathbf{x}(n) - \sum_{k=1}^{j-1} \mathbf{w}_k(n) y_k(n), \ j = 1, 2, ..., l$$

For j = 1:  $x \prime (n) = x (n)$ Evaluation of 1<sup>st</sup> Principal Component  $y_1(n)$ 

For j = 2:  $x'(n) = x(n) - w_1(n) y_1(n)$ Evaluation of  $2^{nd}$  Principal Component  $y_2(n)$  as  $1^{st}$  component after subtracting  $y_1(n)$ For j = 3:  $x'(n) = x(n) - w_1(n) y_1(n) - w_2(n) y_2(n)$ Evaluation of  $3^{rd}$  Principal Component  $y_3(n)$  as  $1^{st}$  component after subtracting  $y_1(n)$ and  $y_2(n)$ 

The *l* most significant *Principal Components* correspond to the eigenvectors  $\mathbf{q}_k$  of the *Correlation Matrix*  $\mathbf{R} = E[\mathbf{X}\mathbf{X}^T]$ , k = 1, 2, ..., l oredered by decreasing order of eigenvalues  $\lambda_1 > \lambda_2 > ... > \lambda_l$  and provide the estimate  $\hat{\mathbf{x}}(n)$  of input sample element  $\mathbf{x}(n)$  of m > l characteristics

$$\hat{\mathbf{x}}(n) = \sum_{k=1}^{l} y_k(n) \mathbf{q}_k \text{ yia } l < m$$



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Generalized Hebbian Algorithm (GHA): Image Coding Example (1/2)

- Training Sample: 2000 scanned pictures of Lena  $256 \times 256$  pixels, 256 gray levels as in the 1<sup>st</sup> Original Image
- Training Sample Elements: Images segmented to 1024 non-overlapping Blocks of size  $8 \times 8$  pixels: m = 64 features/block
- Every block corresponds to sample input vector of *m* features (pixels) encoded into 256 gray levels (8 bits/pixel):

$$\mathbf{x}(n) = [x_1(n) \ x_2(n) \ \dots \ x_m(n)]^T$$
  $n = 1, 2, \dots, N$ 

- The sample vectors are fed into a Linear Feedforward Network with l = 8 outputs
- The  $m \times l = 64 \times 8$  synaptic weights  $w_{ji}(n)$  converge to 8 Significant Principal Components at its output nodes:

$$y_j(n) = \sum_{i=1}^m w_{ji}(n) x_i(n), \ j = 1, 2, ..., l$$

- The Learning Rate) is set to  $\eta = 10^{-4}$
- The weights after convergence are depicted in the  $2^{nd}$  Image with  $4 \times 2 = 8$  regions (masks), 64 segments/mask, total  $64 \times 8 = 1024$  segments representing the contribution of 64 features of the sample input to the 8 outputs. White color signifies positive contribution, black negative and gray no contribution
- The **3<sup>rd</sup>** *Image* is a reconstruction of the **Original** using only l = 8 most significant **Principal Components**

$$\hat{\mathbf{x}}(n) = \sum_{k=1}^{n} y_k(n) \, \mathbf{q}_k \,, \qquad \mathbf{q}_k = \lim_{n} \mathbf{w}_k(n) \,, \ \mathbf{w}_k(n) = \left[ w_{k1}(n) \, w_{k2}(n) \,... \, w_{km}(n) \right]^{\mathrm{T}}$$

• The **4**<sup>th</sup> *Image* is a compressed version of the **3**<sup>rd</sup> *Image* with quantized values according to the logarithm of the 8 output variances (*final compression* 11:1)

Original Image



Weights



Using First 8 Components



11 to 1 compression



**Generalized Hebbian Algorithm (GHA): Image Coding Example (2/2)** Original Image (Peppers): 256 × 256 pixels (features), 256 gray levels

**Original Image** 



Weights



Using First 8 Components



12 to 1 compression via quantization of weights into 8 **Significant Principal Components** determined for **Peppers** 



Weights



12 to 1 compression via quantization of weights into 8 **Significant Principal Components** determined for **Lena** but applied for **Peppers** (**GENERALIZATION ?**)



Weights



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Overview of Self-Organizing Maps (SOM)

- Self-Organized Maps (SOM) refer to *nonlinear* Neural Networks with *m*-dimensional inputs  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$  (Sample Vectors, Examples)
- By analogy to mammal brain functions, *Kohonen* suggested in 1982 a *Feedforward Neural Network* of a Single Layer of neurons j = 1, 2, ..., l placed in a *Feature Map Lattice*
- Input nodes interact with *postsynaptic* lattice neurons with weights  $\mathbf{w}_{j} = \begin{bmatrix} w_{j1} & w_{j2} & \dots & w_{jm} \end{bmatrix}^{T}$
- The activated states of *postsynaptic* neurons reflect the system estimate for  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$  regarding the closest resemblance to *patterns* stored in SOM regions
- Regions are neighborhoods of active neurons, determined via Competitive Unsupervised Learning around the closest neuron (winner) to input vector x

### **SOM Applications**

- Selection of dominant features in multi-dimensional sample spaces
- Image compression by identification of similar regions
- Pattern recognition, classification of images
- Reconstruction of images, filtering of interference and noise
- Completion of partially damaged examples



### SOM Configuration (1/5)

- Algorithm of *feature map* configuration is based on *Hebb* principles for topological selforganization of neural nets into *postsynaptic neuron grids* (arrays, lattices)
- It saves via unsupervised learning *patterns* in the training data by selscting  $l \ll m$  features (*data compression, dimensionality reduction*)
- After configuration **SOM** attempts to reconstruct incomplete or distorted due to noise new examples, based on statistical similarity to pre-stored patterns
- Comparison with K-Means Clustering: With an adequate number of neurons SOM also identifies K as the number of winning neurons, without the need for repeated trials!



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING SOM Configuration (2/5)

- An unsupervised learning algorithm identifies the closest neuron (winning neuron) for every input training vector x via a **Competition Process** that maximizes a discriminant function
- The winner determines a region of active neurons via a **Cooperation Process** with postsynaptic neighbors in the two-dimensional array, resulting in topological *feature map* of active neurons via *self-organization*
- In the Hebbian self-organization the weight vector w<sub>j</sub> is updated with each training input vector x. For stability of the iterative learning an Adaptive Process may guarantee that weights do not increase in uncontrollable fashion



### STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING SOM Configuration (3/5)

### **Competition Process**

During training it identifies the closest postsynaptic neuron j (winner) for every input x via competition that maximizes a discriminant function of the inner product  $w_i^T x$ 

• 
$$\forall \mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$$
 select  $\mathbf{w}_j = [w_{j1} \ w_{j2} \dots \ w_{jm}]^T$  for postsynaptic neurons  $j = 1, 2, \dots, l$ 

• Select winning neuron  $i(\mathbf{x})$  as the one with the maximum  $\mathbf{w}_j^T \mathbf{x}$  (the activation center within the array). Its selection is equivalent to identifying the minimum Euclidean distance between vectors  $\mathbf{x}$  and  $\mathbf{w}_j$ . If  $\|\mathbf{w}_j\| = 1$ :



### STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING SOM Configuration (4/5)

### **Cooperation Process**

- During training, the winning neuron  $i(\mathbf{x})$  defines a region  $h_{j,i(\mathbf{x})}$  of activated neighbors within a lateral distance  $d_{j,i(\mathbf{x})}$
- A usual choice: Gaussian Function  $h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i(\mathbf{x})}^2}{2\sigma^2}\right)$
- The standard deviation  $\sigma$  may decrease as training proceeds, attenuating spatial correlations of nodes in the array and accelerating convergence of synaptic weights



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING SOM Configuration (5/5)

### **Adaptive Process**

The *Hebbian* based self-organized learning algorithm requires corrections while iterating towards a finite weight vector  $\mathbf{w}_j$  for each example  $\mathbf{x}$ . A remedy is to adjust its coordinates via a *forgetting term* proportional to the output in each iteration

- Definition of Forgetting Term: g (y<sub>i</sub>) w<sub>j</sub> with g (y<sub>i</sub>) a non-negative function of output y<sub>i</sub> with g(y<sub>i</sub>) = 0 for y<sub>i</sub> = 0
- Updates are guided by differences  $\Delta \mathbf{w}_{i} = \eta y_{i} \mathbf{x} g(y_{i}) \mathbf{w}_{j}$  where:

η: learning hyperparameter,  $y_i$ **x**: Hebbian term  $g(y_i)$  **w**<sub>j</sub>: forgetting term

- With linear forgetting term  $g(y_i) = \eta y_i$  and  $y_i = h_{j,i(\mathbf{x})}$  we obtain:  $\Delta \mathbf{w}_j = \eta h_{j,i(\mathbf{x})}(\mathbf{x}-\mathbf{w}_j)$  with  $i(\mathbf{x})$  the winning neuron for input  $\mathbf{x}$
- In iteration  $n \rightarrow n + 1$  and with decreasing hyperparameter  $\eta(n)$ :

$$\mathbf{w}_{j}(n+1) = \mathbf{w}_{j}(n) + \eta(n)h_{j,i(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_{j}(n))$$

The synaptic weights to the winning neuron converge to input sample vector  $\mathbf{x}$  and those of the neighboring neurons reflect the distribution of the training sample

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Autoencoders (1/2)

### Encoder:

Input  $\mathbf{x} = [x_1 \ x_2 \dots \ x_m]^T$ Output  $\mathbf{h} = \mathbf{F}_{\mathbf{e}}(\mathbf{x}) = [h_1 \ h_2 \dots \ h_l]^T$ ,  $l \ll m$  (code, latent variables)

### Decoder:

Input  $\mathbf{h} = [h_1 \ h_2 \dots \ h_l]^T$ Output  $\mathbf{x'} = \mathbf{F_d}(\mathbf{h}) = [x_1' \ x_2' \dots \ x_m']^T$  (reconstruction of x)

### Bottleneck:

Middle layer of hidden nodes reflecting *l* latent variables

# Unsupervised Learning Algorithm:

MSE minimization  $\|\mathbf{x} - \mathbf{x'}\|^2$  via backpropagation for unlabeled training sample elements



The *decoder* layers are not used **after** the *encoder* parameter tuning (training) stage for applications relying on determination of *latent variables*, e.g. for dimensionality reduction and compression

- The latent variables reveal reduced feature maps. With linear neural nets and zero bias they estimate the *l* Principal Components of unlabeled sample sets
- Applications include image compression, pattern recognition, correction and completion of distorted (noisy) images, anomaly detection from unlabeled training datasets

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING Autoencoders (2/2)

### **Autoencoder Use for Anomaly Detection**

- Used to identify **outliers**, rare invalid *anomalies* hidden among many *normal unlabeled* sample elements
- The autoencoder parameters in all stages are tuned by back-propagation with normal examples as input training elements
- Close *reconstruction* of input elements at the output layer is considered as the test for a test (or new) element to be classified as normal or dismissed as a statistical outlier
- Critical hyperparameters: *Reconstruction deviation* (e.g. MSE) and *threshold* classifying an element as *statistically normal* or as an *outlier*



https://saketsathe.net/downloads/autoencode.pdf

**Note**: All layers of the Autoencoder are employed in the post-training phase of its operation as Anomaly Detector