ε.δε.μ²
ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Overview of Neural Networks
Hebb's Rule
Weight Tuning via Supervised Learning
Back-Propagation Algorithm

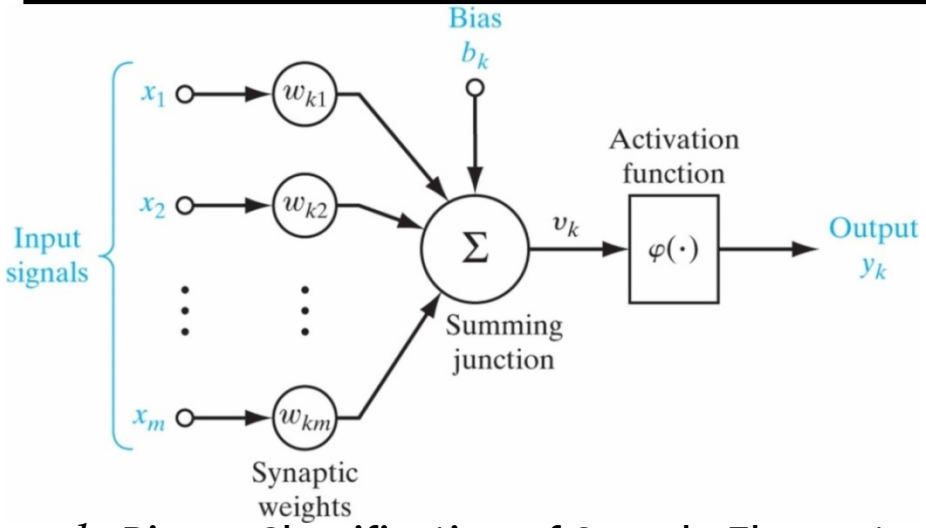
Prof. Vasilis Maglaris
maglaris@netmode.ntua.gr
www.netmode.ntua.gr
Room 002, New ECE Building

Tuesday February 18, 2025

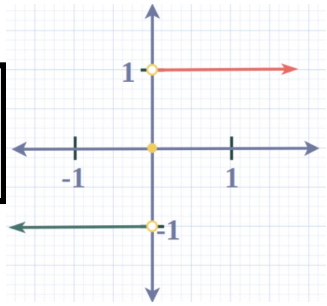
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Mnon-linear Model of Binary Artificial Neuron, *Rosemblatt's* Perceptron

McCulloch & ***Pitts*** (1943): Neural Nets (NNs) in Machine Learning (ML)
Hebb (1949): Self-organizing learning principles
Rosemblatt (1958): Supervised Learning, Perceptron
Rumelhart (1985): Back Propagation Algorithm



Signum Function - sgn
$\varphi(v) \in \{-1, +1\}$



Neuron k: Binary Classification of Sample Elements $\mathbf{x} = [x_1\ x_2 \ldots x_m]^{\mathrm{T}}$, classes $\{-1,\ +1\}$

Input Signals: $x_j \triangleq \pm 1,\ j = 1,2,\ldots,m$ (Vector $\mathbf{x}$ of m Binary Features)

Synaptic Weights: $\mathbf{w}_k = [w_{k0}\ w_{k1} \ldots w_{km}]^{\mathrm{T}}$

Bias: $b_k \triangleq w_{k0}$ (Intercept term $x_0 \triangleq +1$)

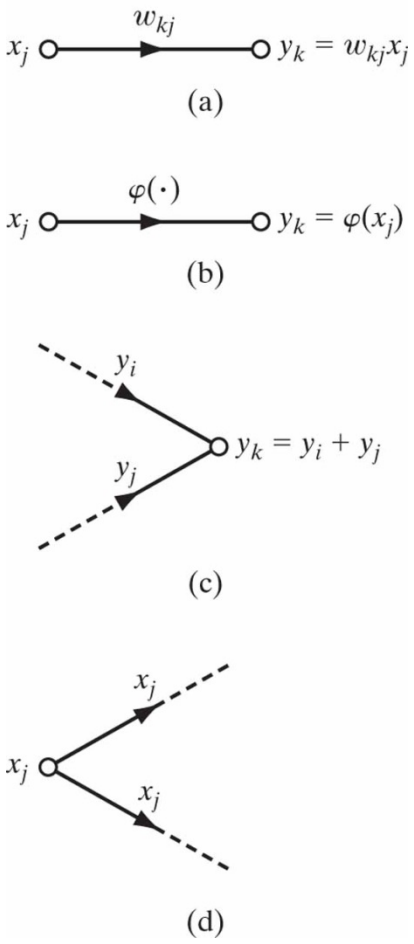
Induced Local Field - Activation Potential: $v_k = \sum_{j=0}^{m} w_{kj} x_j + b_k$

Binary Output: $y_k = \varphi(v_k) = \mathrm{sgn}(v_k) \in \{-1(\textit{Inactive}),\ +1(\textit{Active})\}$

Supervised Learning: Tuning of w_{kj} based on N labeled training elements $\{\mathbf{x}(n), d(n)\}$ to minimize deviations e.g. Mean Square Error (***MSE***): $\min_{\mathbf{w}_k} \{\frac{1}{N} \sum_{n=1}^{N} [d(n) - y_k(n)]^2\}$

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

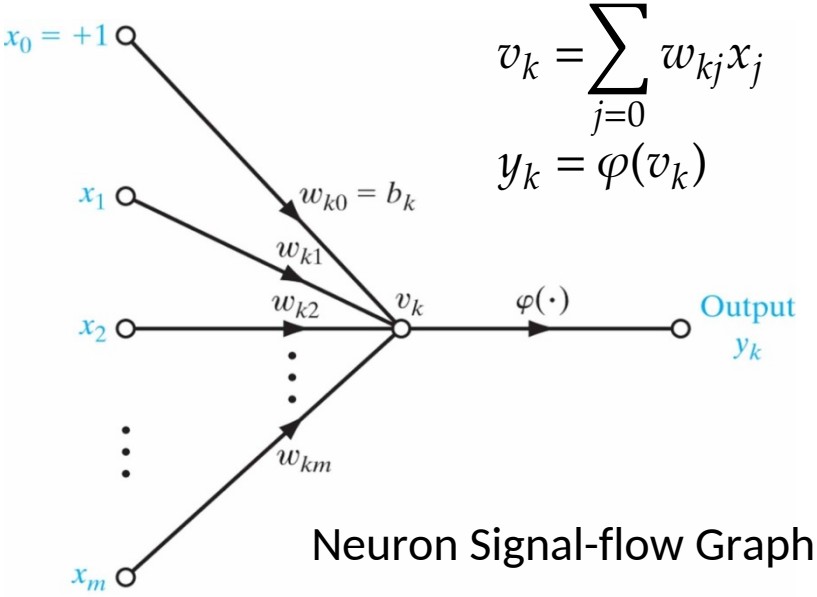
Neural Networks as Directed Graphs



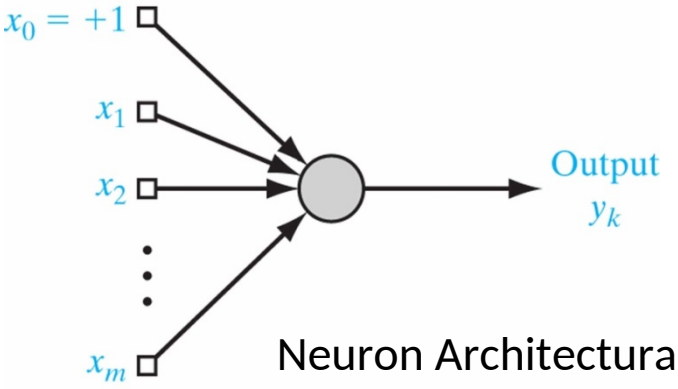
Signal-flow Graph Rules

$$v_k = \sum_{j=0}^{m} w_{kj}x_j$$

$$y_k = \varphi(v_k)$$



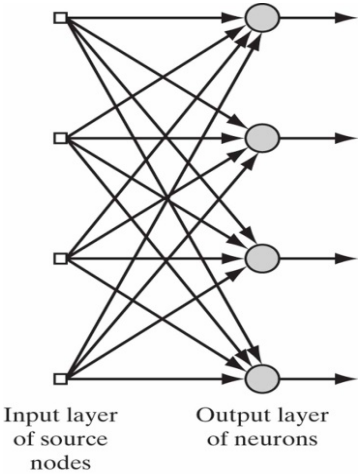
Neuron Signal-flow Graph



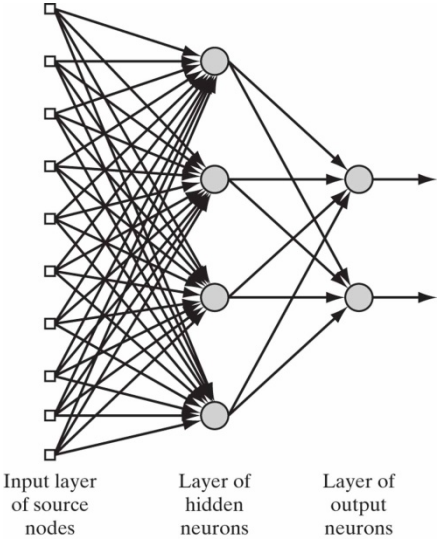
Neuron Architectural Graph

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

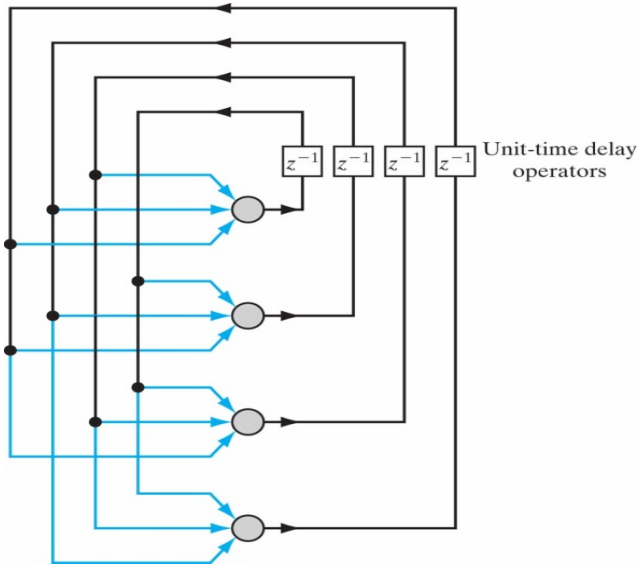
Μοντέλα Νευρωνικών ΔικτύωνNeural Network Models



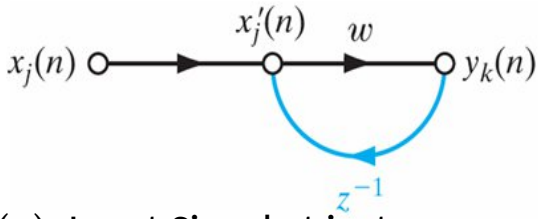
Single-Layer Feedforward Network



Multilayer Feedforward Network with Hidden Neurons



Recurrent Network with no Hidden Neurons



$x_j(n)$: Input Signal at instance n

$y_k(n)$: Output Signal at instance n

$x_j\prime(n)$: Internal Input Signal at instance n

$$y_k(n) = \sum_{l=0}^{\infty} w^{l+1} x_j(n-l)$$

Signal-flow Graph, Infinite Impulse Response IIR Filter with Feedback

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Design Phases of Neural Network (NN)

❖ ***Supervised Learning*** using *labeled training sample points* selected from the environment:
- Evaluation (tuning) of parameters (*synaptic weights* w_{kj}, *biases* b_k) of NN model
- Iterative training algorithms (e.g. minimizing *Mean Square Error* - *MSE*)

❖ ***Validation*** of NN model ability via additional known *validation sample points*:
- Selection of *hyperparameters* (number of neurons, layers, convergence criterion for parameter tuning…)
- Avoidance of *overfitting*

❖ ***Testing*** of selected model accuracy in predicting new *testing sample points*:
- Checking the capability of selected NN model to *generalize* if fed with new data points of the same environment, statistically similar but not used for training (and validation)
- Last design phase prior to *production deployment*

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Knowledge Representation

The discrimination capacity (prediction, perception, desired response) of the NN model to external environment inputs depends on using:

- **Prior Information** of the environment attributes
- **Observations** obtained by extensive (and selective) monitoring the environment
  - Usually tempered by noisy sensor errors
  - They provide ***training sample elements*** for NN parameter tuning, usually after filtering and pre-normalization. They may be:
    - **Labeled,** with appended output desired characteristics (additional features) known to a *supervisor* that guides NN parameter tuning convergence
    - **Unlabeled**, without indication of outut characteristics but providing the NN with a sample for the environment statistics for *unsupervised* parameter tuning

Rule of *Hebb*, 1949: Inspired by Neuro-Physiological Learning

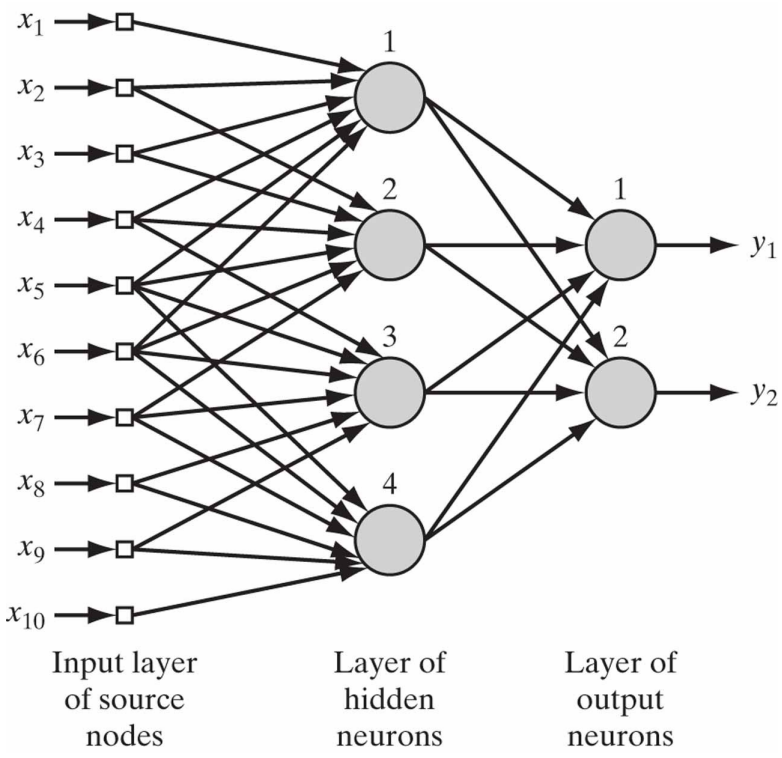
In artificial neural networks of binary state neurons, synapses between active neurons exhibit tendencies of enforcement similarly to neuro-physiological learning systems

Synaptic weights w_{ij} between active neurons i,j tend to increase, while others tend to zero. This rule guides self-tuning of complex Machine Learning (ML) systems, e.g. *Unsupervised Learning* in Self-Organizing Maps, Boltzmann Machines e.t.c.

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Building Prior Information into Neural Network Design

Architectural Choices guided by Prior Information - Convolutional NN's



Convolutional Network Simplification Example

- Feedforward Fully Connected NN with *Input Layer* of 10 source nodes (features), *Hidden Layer* of 4 nodes and *Output Layer* of 2 nodes
- The hidden neurons are fed only from 6 input nodes, subsets of the 10 inputs, that comprise their *receptive fields*. Their *Activation Potential* (or *Induced Local Field*) v_j is the weighted sum of 6 out of 10 inputs
- Weights w_i, $i = 1,\ldots,6$ between inputs x_i, $i = 1,\ldots,10$ and hidden nodes $1,2,3,4$ are commonly shared (*weight sharing*). Their *Activation Potential* is assumed to be the *Convolutional Sums*:

$$v_j = \sum_{i=1}^{6} w_i x_{i+j-1}, \quad j = 1,2,3,4$$

thus

$$v_1 = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$
$$v_2 = w_1x_2 + w_2x_3 + w_3x_4 + w_4x_5 + w_5x_6 + w_6x_7$$
$$v_3 = w_1x_4 + w_2x_5 + w_3x_6 + w_4x_7 + w_5x_8 + w_6x_9$$
$$v_4 = w_1x_5 + w_2x_6 + w_3x_7 + w_4x_8 + w_5x_9 + w_6x_{10}$$

Learning for w_i from a reduced number of input features i, equally fed to all hidden nodes, is significantly lighter than evaluating all synaptic weights for fully connected NN's. It is justified when prior information points to uniform selection of input features, e.g. computer vision models for restricted retina reception

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Review of NN Design Principles

Architectural Choice

- Layers: Input, output, hidden
- Feedforward or recurrent model with feedback

Training for Parameter Tuning

- **Supervised Learning**
  - Use of **Labeled Training Sample Points** (input – output pairs) to achieve minimization of prediction error
- **Unlabeled Training Sample Points**
  - **Unsupervised Learning** and **Reinforcement Learning**
- **Normalization of Training Datasets**
  - **Min-max Normalization** of training sample features, prior to learning
- **Iteration Schemas of Training Algorithm**
  - **On-line Training** proceeding step-by-step upon each training sample point application in a random order (**Stochastic Gradient Descent**)
  - **Batch Training** proceeding in each iteration after application of the entire training sample (**Batch Gradient Descent**)
  - **Mini-Batch Training** by applying in each iteration subsets (mini batches) of the training sample

Epochs are periods of repeated application of the entire training sample, usually with randomized re-selection of the order of appearance of its elements
https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

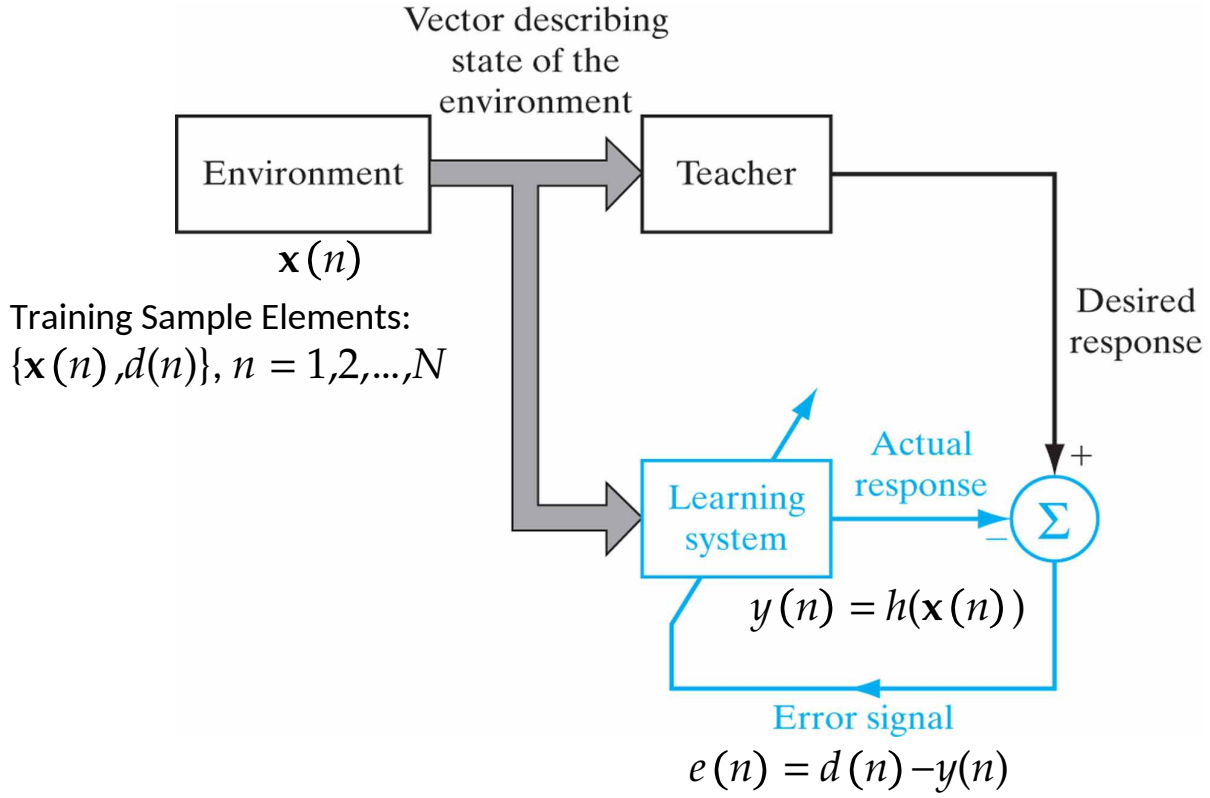
Learning System (Neural Net)

- Estimation of (scalar) output $y(n) = h(\mathbf{x}(n))$ of the n^{th} ***Environment State Vector*** input e.g. binary classification based on learned statistics of input state coordinates (features)

Training Phase

- Proceeds with a ***Labeled Training Sample*** of N elements (examples) that describe instances of the state of the environment $\mathbf{x}(n)$, along with $d(n)$, the desired response or label known to a ***Teacher***
- For each training sample element the ***Learning System*** (Neural Network) evaluates an ***Actual Response*** (output) $y(n) = h(\mathbf{x}(n))$
- The system computes the ***Error Signal*** $e(n) = d(n) - y(n)$ i.e. the deviation of its actual response from the ***Desired Response*** conveyed by the ***Teacher*** (supervisor)
- The parameters of $h(.)$ are tuned in N iterations to minimize an ***Error***

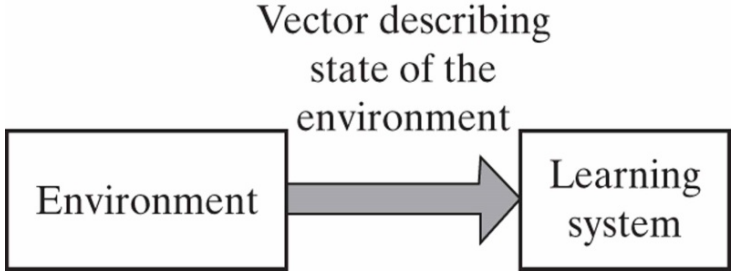
Supervised Learning



The algorithm usually proceeds with randomly selected order of the N training sample element inputs, towards the gradient of the error function (e.g. *Stochastic Gradient Descent*)

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Unsupervised Learning

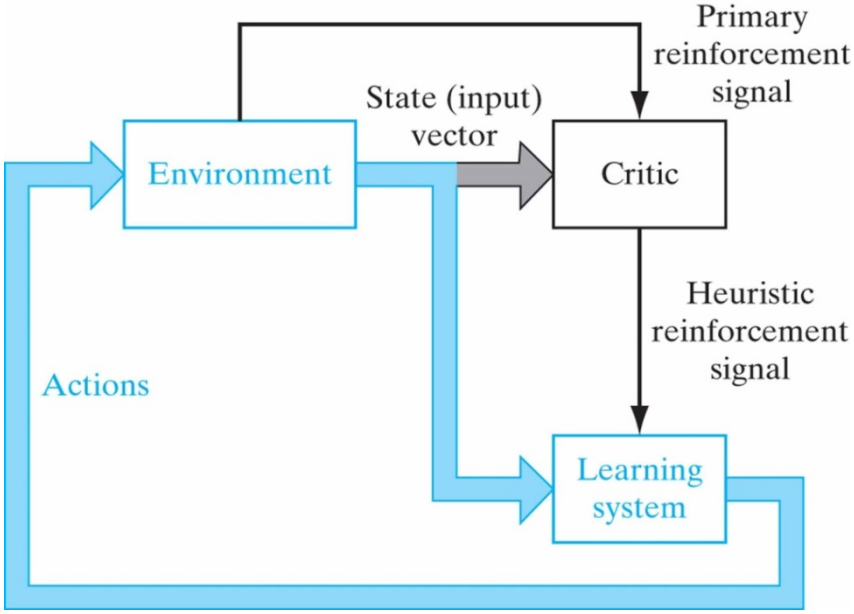


Unsupervised learning does no rely on labeled data but, compared to supervised learning algorithms, requires significantly larger training datasets and may exhibit reliability problems

- The ***Learning System*** is self-tuned without external supervision by discovering important statistical properties exhibited by the training elements (***state vectors***) of the ***Environment***
- In its training phase it infers ***stochastic features*** and ***patterns*** of large ***unlabeled training datasets*** that point to models, processing, storage and classification methods e.g. by ***clustering*** sample elements
- The learning system can ***generate sample elements***, conforming to the environment statistics. As a result, it can be used to classify and complement noisy and/or incomplete data items (e.g. in image processing and pattern recognition)
- Uncontrollable guessing of statistical properties in training data and a limited understanding of important features of the environment, may lead to severe ***overfitting*** (excessive reliance to insignificant ***outliers***) and to wrong classification or hallucinations
- An ***unsupervised learning*** example: A Neural Network consisting of input nodes and a dense layer of hidden nodes competing on encoding salient features of input data as in ***Self-Organized Maps*** (***SOM***)
- Potential use of the ***Rule*** of ***Hebb***: During (unsupervised) training select for activation only neurons with the maximum activation potential v_k (***winner-takes-all***)

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Reinforcement Learning



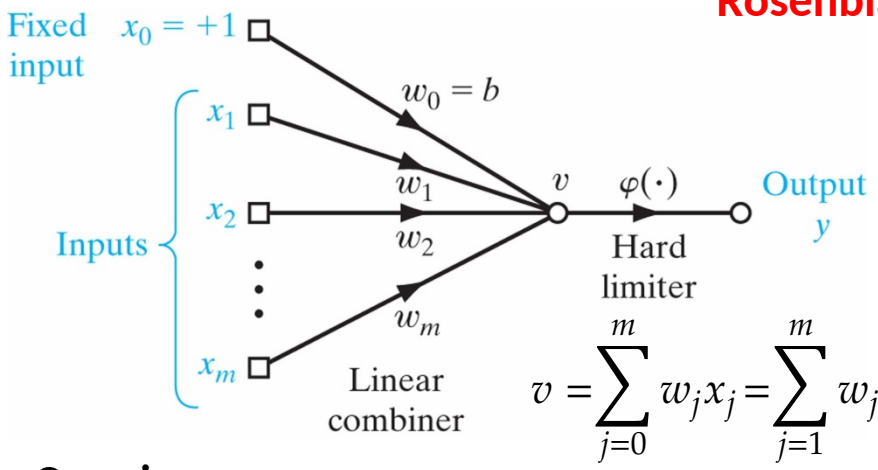
It follows the ***Dynamic Programming*** - ***Optimal Stochastic Control*** paradigm. The learning system parameters may be tuned by dynamically exploiting and exploring candidate ***Environmental State Trajectories***, anticipated from prior knowledge or simulated during a training phase. Control ***Actions*** aim to maximize (minimize) a long-term expected reward (cost), accumulated along ***Ttrajectories***

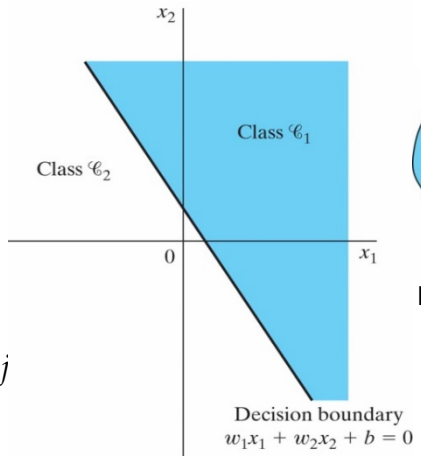
- The ***Learning System*** learns possible ***trajectories*** of the ***Environment State Vector*** $\mathbf{x}(n)$ from ***prior information*** and/or ***unlabeled training data***. Instead of a direct supervisor. It proceeds by taking ***Control Actions*** considering additional ***Reinforcement Signals*** from an external ***Critic*** or ***Agent***
- The ***Learning System Actions*** affect the ***Environment State Transitions***, thus govern its evolution. In most cases transition probabilities - actions are assumed to abide by ***Markov Decision Process*** models
- In every iteration, the ***Environment*** computes a scalar ***Primary Reinforcement Signal*** that is conveyed to the ***Critic*** along with the current state vector
- The ***Critic*** evaluates state – action ***policy*** pairs by considering mid or long-term cost/reward objectives resulting from expected state evolution (***trajectory***) of candidate policies
- A simplified ***Heuristic Reinforcement Signal*** is conveyed to the ***Learning System*** which triggers appropriate ***Control Actions***, fed-back to the ***Environment*** to guide its long-term evolution

The Chinese ***DeepSeek*** Chat box employs a variant of ***Reinforcement Learning***, unlike the OpenAI ***ChatGPT*** which is based on massive ***Supervised Learning*** Large Language Models (***LLM***)

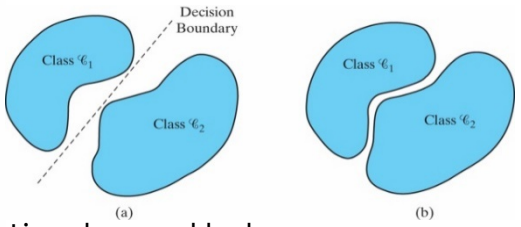
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Rosenblatt's Perceptron





Non-linearly separable classes



Linearly separable classes

Overview:

Rosenblatt introduced the ***Single-Layer Perceptron*** as a neuron of ***linear*** **Induced Local Field** v and ***non-linear*** **Activation Function** $\varphi(v)$ (***Threshold Function***, ***Hard Limiter*** or ***Signum Function***) for binary classification of sample elements $\mathbf{x} = [x_0 \; x_1 ... x_m]^{\mathrm{T}}$ into two ***linearly separable*** classes:

$$\mathscr{C}_1 \text{ if } y = \varphi(v) = 1, \;\; \mathscr{C}_2 \text{ if } y = \varphi(v) = 0 \text{ or if } \; y = \varphi(v) = -1$$

Synaptic weights $\mathbf{w} = [w_0 \; w_1 ... w_m]^{\mathrm{T}}$ are tuned on-line (***stochastic iterative method***) via an error-correction algorithm on labeled training sample elements $\{\mathbf{x}(n), d(n)\}$, $n = 1,2,...,N$ via ***supervised learning*** to minimize an error function (e.g. ***MSE***) of ***deviations*** $[d(n) - y(n)]$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta [d(n) - y(n)] \mathbf{x}(n)$$

If the ***learning-rate hyperparameter*** η, $0 < \eta \leq 1$ is small it usually leads to (slow) convergence. If it is large it may lead to fast convergence (e.g. for environments of significant element deviations) but may skip optimality dua to oscillations)

Note: With Gaussian elements $\mathbf{x}(n)$ ***Bayes Classifiers*** into two classes $\mathscr{C}_1$, $\mathscr{C}_2$ (based on minimization of error probability with a-known a-priori probabilities p_1,p_2) is identical to the ***Rosenblatt Perceptron***

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

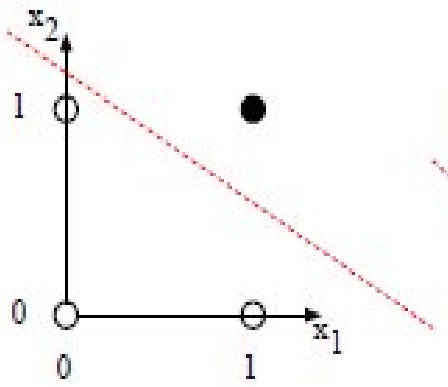
Perceptron Classification of Boolean Operations: Functions AND, OR (*XOR* ?)

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

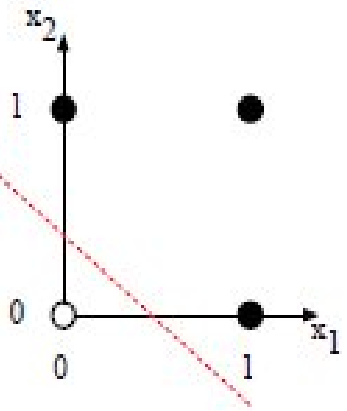
Threshold Activation Function:

$$y = h(v) = \begin{cases} 0, & v \le 0 \\ 1, & v > 0 \end{cases}$$

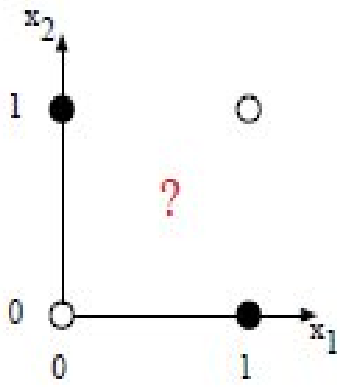
$$y = \begin{cases} 0, & w_1 \cdot x_1 + w_2 \cdot x_2 + b \le 0 \\ 1, & w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0 \end{cases}$$



a) x_1 AND x_2

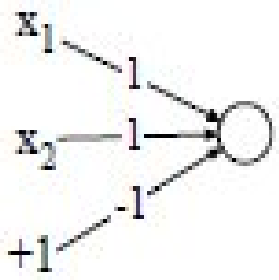


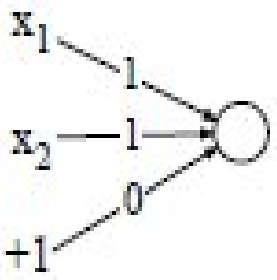
b) x_1 OR x_2



c) x_1 XOR x_2

Linearly Separable Boolean Functions AND, OR





Non-Linear Separable Boolean Function XOR

(Single Layer Perceptron not capable to classify y)

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

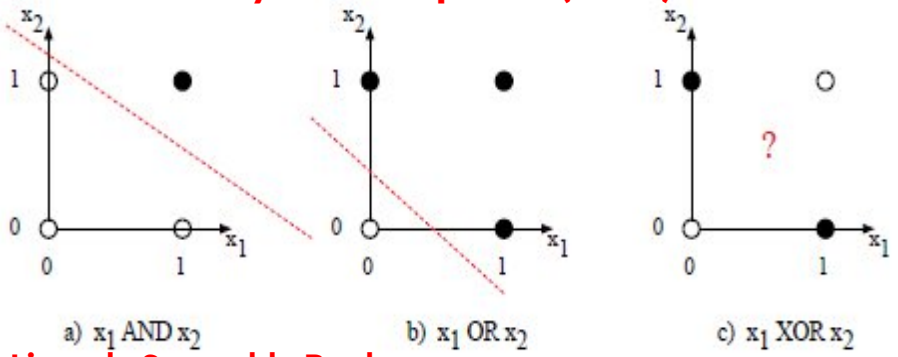
Perceptron Classification of Boolean Operations: Function *XOR via* Feedforward Multi-Layer Perceptron (MLP)

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

Threshold Activation Function:

$$y = h(v) = \begin{cases} 0, & v \le 0 \\ 1, & v > 0 \end{cases}$$

$$y = \begin{cases} 0, & w_1 \cdot x_1 + w_2 \cdot x_2 + b \le 0 \\ 1, & w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0 \end{cases}$$

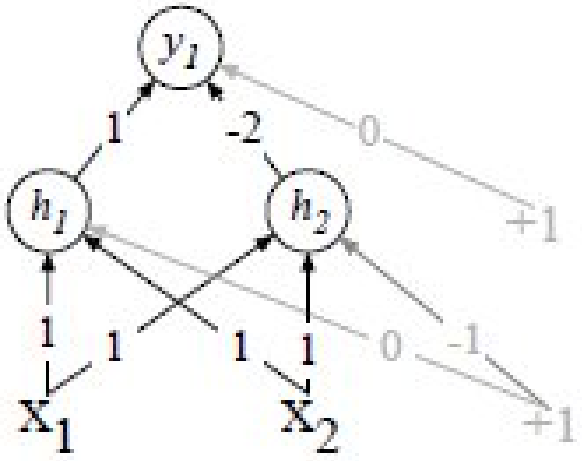


a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

Linearly Separable Boolean Functions AND, OR

Non-Linearly Separable Boolean Function XOR
(Single Layer Perceptron not capable to classify y)

XOR: Implementation via Hidden Neurons h_1, h_2 → **Multi-Layer Perceptron, MLP**



Output of Neuron h_1: $a_1 = h(w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1)$

Output of Neuron h_2: $a_2 = h(w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2)$

Output of Neuron y_1: $y = h(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3)$

$$w_{11} = w_{12} = w_{21} = w_{22} = 1,\ b_1 = 0,\ \ b_2 = -1$$

$$w_{13} = 1, w_{23} = -2,\ b_3 = 0$$

x_1: 0 0 1 1
x_2: 0 1 0 1
a_1: 0 1 1 0
a_2: 0 0 1 0
y: 0 1 1 0

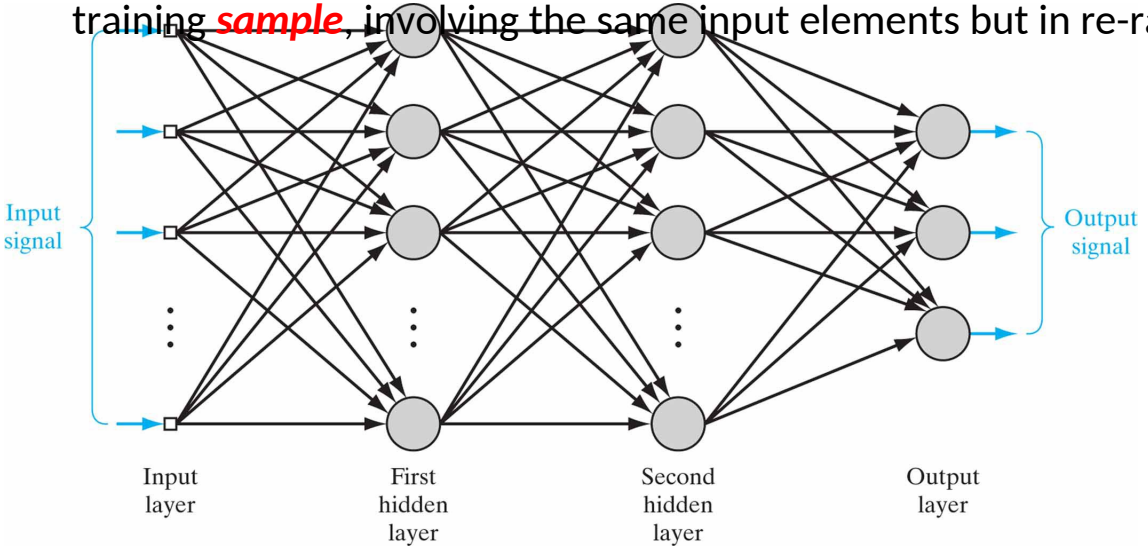
Daniel Jurafsky, James H. Martin, **"Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition"**, Third Edition draft, 2025

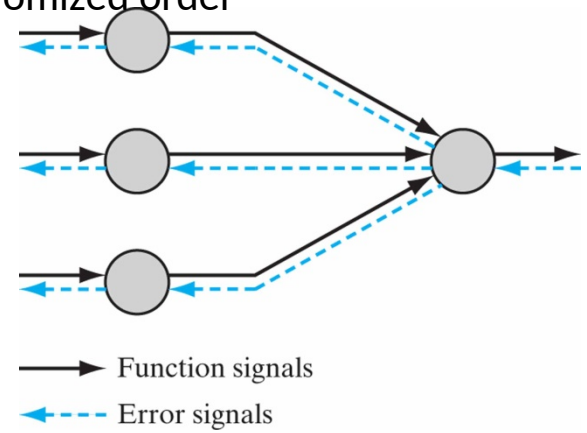
STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Multilayer Perceptron, Back-Propagation Algorithm (1/3)

Algorithm Summary:

1. Assumption: ***Differentiable*** non-linear activation function $\varphi_j(v_j)$ of neuron j (e.g. ***Logistic Function***)
2. Dense ***layers*** of ***hidden*** neurons with weights w_{ji} (from $i \to j$). Hidden neuron state values reflect ***features*** of the input training sample, tuned via ***supervised learning***
3. Weights w_{ji} are tuned via the ***Back-Propagation Algorithm***, usually ***on-line*** (***stochastic iterative method***) in randomized order for each ***labeled*** learning input instance $\{\mathbf{x}(n), d_j(n)\}$, $n = 1,2,...,N$ in two phases per iteration n:
   i. **Forward Phase**: Input $\mathbf{x}(n)$ traverses the network via ***Forward Function Signals*** with weights $w_{ji}(n)$ as determined up to this point, and evaluates the temporary state value $y_j(n)$ of the ***Output*** neuron j
   ii. **Backward Phase**: Deviations $e_j(n) = d_j(n) - y_j(n)$ traverse the network as ***Error Signals*** and correct synaptic weights
4. Final convergence is declared after several ***epochs*** that involve two-phase iterations for the entire training ***sample***, involving the same input elements but in re-randomized order





STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Multilayer Perceptron, Back-Propagation Algorithm (2/3)

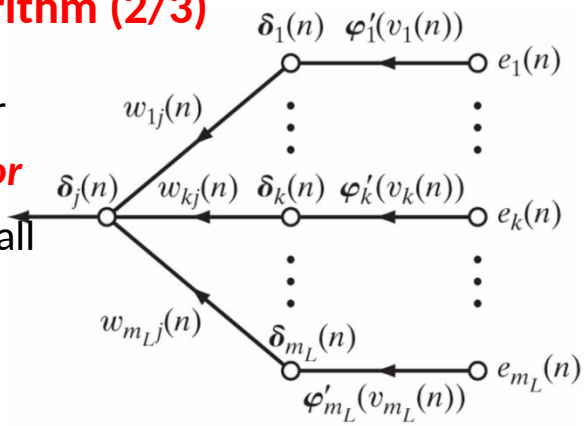
Least Mean Square – LMS per Epoch Convergence:

For the training sample-element (example, instance) $\{\mathbf{x}(n), d_j(n)\}$ the error signal at the j^{th} output node is $e_j(n) = d_j(n) - y_j(n)$. The ***Mean Square Error*** (***MSE***) of all temporary deviations is $\mathscr{E}(n) = \frac{1}{2}\sum_j e_j^2(n)$ and the average for all N training sample-elements in and ***Epoch*** is $\mathscr{E}_{AVG}(N)$:

$$\mathscr{E}_{AVG}(N) = \frac{1}{N}\sum_{n=1}^{N}\mathscr{E}(n) = \frac{1}{N}\sum_{n=1}^{N}\left\{\frac{1}{2}\sum_j e_j^2(n)\right\}$$

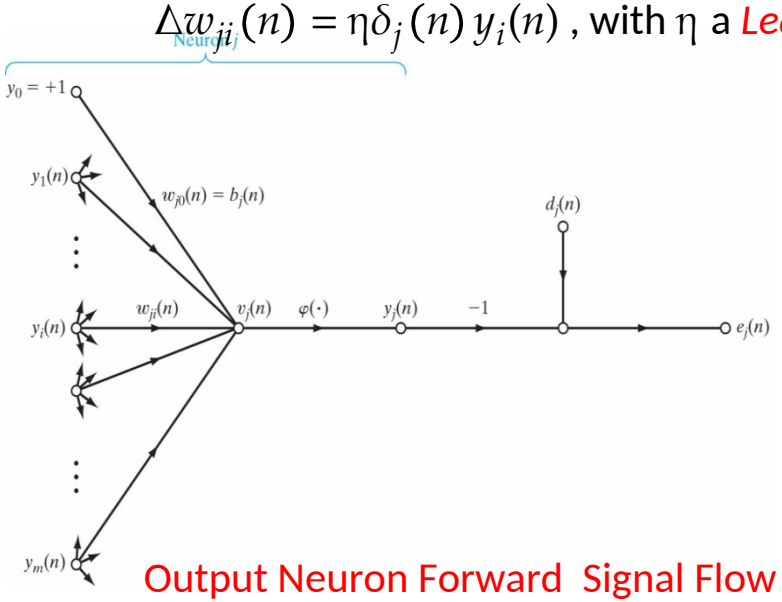
Iterative corrections $\Delta w_{ji}(n)$ of weights $w_{ji}(n)$ lead $\mathscr{E}(n)$ to reduction towards the *local gradient* $\delta_j(n) = \frac{\partial \mathscr{E}(n)}{\partial v_j(n)}$, with $v_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n)$ the *Induced Local Field* of neuron j:

$$\Delta w_{ji}(n) = \eta \delta_j(n)\, y_i(n)$$, with η a *Learning Hyperparameter*

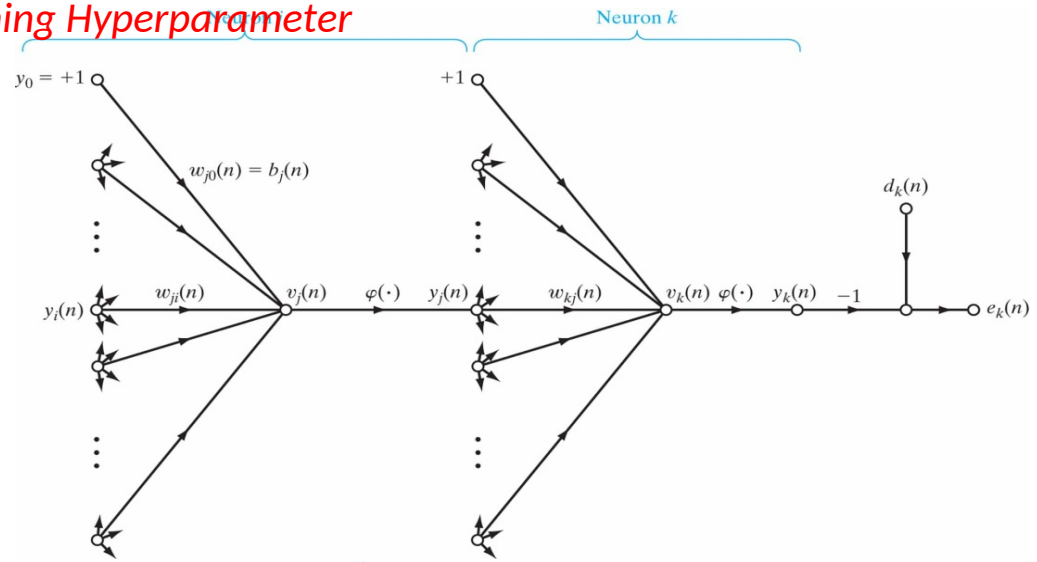


$$\varphi_j'(v_j(n)) = \frac{\partial y_j(n)}{\partial v_j(n)}$$

Backward Signal Flow



Output Neuron Forward Signal Flow



Output Neuron k Signal Flow towards Hidden Neuron j

STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

Multilayer Perceptron, Back-Propagation Algorithm (3/3)

Steps of On-line Learning Algorithm

1. For each training sample-element $\{\mathbf{x}(n), d_j(n)\}$, $n = 1,2,...,N$ initialize parameters (weights) $w_{ji}^{(l)}(0)$ for all layers $l = 0,1,2,...,L$
2. Proceed with ***forward phase*** calculations:
   - For each neuron j and each layer $l = 1,2,..., L$ the ***induced local field*** is:
   $$v_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) y_j^{(l-1)}(n)$$
   - For $i = 0$, $y_0^{(l-1)} = +1, w_{j0}^{(l)} = b_j^{(l)}(n)$ (bias into neuron j)
   - The output value of neuron j at layer l is $y_j^{(l)}(n) = \varphi_j(v_j^{(l)}(n))$
   - For the first hidden layer $l = 0$: $y_j^{(0)}(n) = x_j(n)$
   - If j belongs to the output layer $l = L$: $y_j^{(L)}(n) = o_j(n)$, the final output j of the ***Multilayer Perceptron***. The error signal is $e_j(n) = d_j(n) - o_j(n)$
3. Continue with ***backward phase*** calculations for $l = L, L-1,..., 0$:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\ \varphi_j'\left(v_j^{(L)}(n)\right) & \text{if } j \text{ belongs to the output layer } L \\ \varphi_j'\left(v_j^{(l)}(n)\right) \sum_k \delta_K^{(l+1)}(n)\, w_{ki}^{(l+1)} & \text{if } j \text{ belongs to the hidden layer } l \end{cases}$$

Weight updates at step $n+1$ proceed in proportion to the ***Learning Hyperparameter*** η. To avoid wild oscillations values of the previous iteration $n-1$ may also be considered weighted by a ***momentum constant*** $\alpha \geq 0$:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha \times w_{ji}^{(l)}(n-1) + \eta \times \delta_j^{(l)}(n)(n) y_j^{(l-1)}(n)$$