ONLY GOD

1402/ 2023

Neural Networks & Deep Learning

NN Learning

CSE & IT Department
ECE School
Shiraz University

mansoori@shirazu.ac.ir

NN Learning

mansoori@shirazu.ac.ir

Training an NN

- Training: procedure of determining appropriate weights

- General procedure: learning appropriate weights from training data

Start with random initial weights

Using training data, adjust weights in small steps

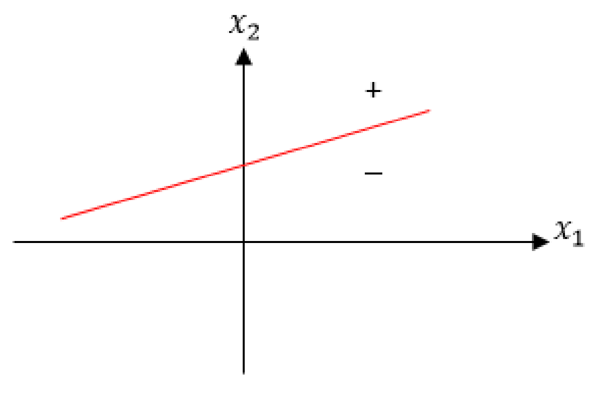
Until required outputs are produced

- Brute force derivation: trainings are iterative learning algorithms

NN Classifier Learning

mansoori@shirazu.ac.ir

- For a single layer NN performing binary classification:
  - Decision boundary is hyperplane

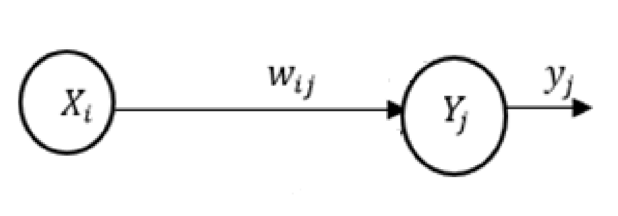


  - Learning is process of shifting around hyperplane until each training pattern is classified correctly

- To formalize process of shifting around into a systematic implementable algorithm:
  - Shifting around is split up into a number of small steps

mansoori@shirazu.ac.ir

Weights Updating



- If network weights are w_{ij}, then shifting process corresponds to moving by Δw_{ij}:

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

- Threshold (bias) is treated as a weight:

$$b_j(new) = b_j(old) + \Delta b_j$$

- Weight changes Δw_{ij} need to be applied repeatedly, for each weight w_{ij} in network, and for each training pattern in training set

End of Learning

- One pass through all weights for whole training set is called one epoch of training
- Eventually, usually after many epochs, when all network outputs match targets for all training patterns, all Δw_{ij} will be zero and process of training will cease (i.e. training process has converged to a solution)

NN Learning

mansoori@shirazu.ac.ir

- Learning is a process by which free parameters of an NN are adapted through a continuing process of stimulation by environment in which network is embedded
- Learning process tries to adapt synaptic weights
- The type of learning is determined by manner in which parameters' changes take place
  - Hebbian, Perceptron, delta, competitive, Boltzmann
  - Supervised, reinforced, unsupervised

Hebbian Learning

mansoori@shirazu.ac.ir

Hebbian Learning

mansoori@shirazu.ac.ir

- In 1949, neuropsychologist Donald Hebb postulated how biological neurons learn:
    - When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells such that A's efficiency as one of the cells firing B, is increased
- There is strong physiological evidence that this type of learning does take place in the region of the brain known as the hippocampus

Hebbian Learning

mansoori@shirazu.ac.ir

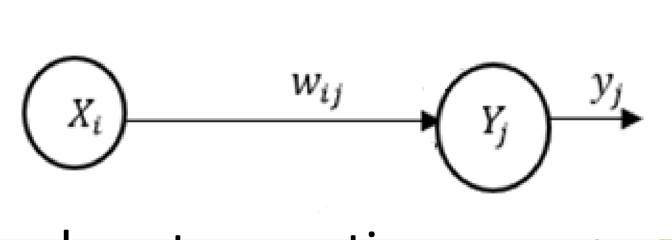
- Hebbian rule is the earliest and simplest learning rule of NNs

- The Hebb learning rule:
  - If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased

- Its supplementary rule:
  - If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated

mansoori@shirazu.ac.ir

Math. Model of Hebbian Learning



- General form of Hebbian rule:

  $$\Delta w_{ij} = F(x_i, y_j)$$

  where $F(.,.)$ is a function of pre-synaptic and post-synaptic activities

- A specific Hebbian rule (activity product rule)

  $$\Delta w_{ij} = \eta\, x_i\, y_j$$ where η: learning rate

  - Drawback: No bounds on increase (or decrease) of w_{ij}

- Generalized activity product rule:

  $$\Delta w_{ij} = \eta\, x_i\, y_j - \alpha\, w_{ij}\, y_j = \alpha\, y_j\, (\kappa\, x_i - w_{ij})$$

  where $\kappa = \frac{\eta}{\alpha}$ and α: positive constant

Math. Model of Hebbian Learning

mansoori@shirazu.ac.ir

- Activity covariance rule:

$$\Delta w_{ij} = \eta \operatorname{cov}(x_i, y_j) = \eta \operatorname{E}[(x_i - \bar{x})(y_j - \bar{y})]$$

where η: proportionality constant

- After simplification:

$$\Delta w_{ij} = \eta \left(\operatorname{E}[x_i \, y_j] - \bar{x}\,\bar{y}\right)$$

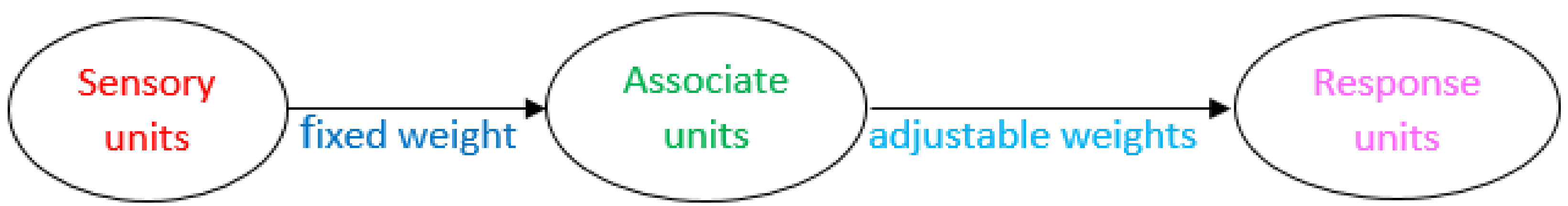
mansoori@shirazu.ac.ir

Perceptron Learning

mansoori@shirazu.ac.ir

Perceptron Learning Rule

- More powerful than Hebb rule
- Is iterative
- Can converge to the correct weights in a finite number of epochs
- The original perceptron (approximation model of retina in visual system) has three layers of neurons



binary random: {+1, 0, -1} binary step

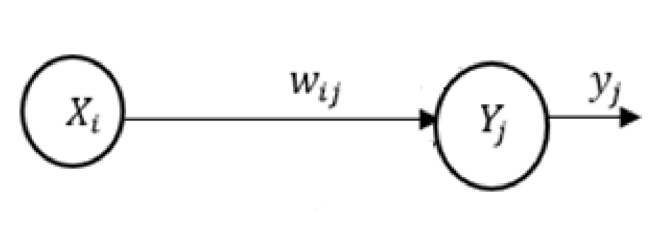
$$f(y\_in) = \begin{cases} 1 & \text{if } \quad y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } \quad y\_in < -\theta \end{cases}$$

If $\theta = 0$ then $f(y\_in) = \text{sgn}(y\_in)$ (θ: fixed)

Perceptron Learning Rule

mansoori@shirazu.ac.ir

- Only if an error occurred for a training pattern, the weight would be adjusted:



$\Delta w_{ij} = \alpha\, x_i\, t_j$ when t_j: binary

α : learning rate

- Training would continue until no error occurred

- When t_j, y_j: bipolar

$$\Delta w_{ij} = \eta\,(t_j - y_j)x_i$$

where $y_j = f(y\_in_j) = \begin{cases} 1 & \text{if } y\_in_j \geq 0 \\ -1 & \text{if } y\_in_j < 0 \end{cases}$

Perceptron Learning Rule

The Perceptron learning rule convergence theorem:

- If there does exist a possible set of weights for a Perceptron which solves the given problem correctly, then the Perceptron learning rule will find them in a finite number of iterations

- Moreover, if a problem is linearly separable, then the Perceptron learning rule will find a set of weights in a finite number of iterations that solves the problem correctly

Perceptron Learning Rule

mansoori@shirazu.ac.ir

The Perceptron learning rule convergence theorem:

- If there is a weight vector $\overrightarrow{w^*}$ such that $f(\vec{x}.\overrightarrow{w^*}) = \vec{t}$ for all training patterns, then for any starting vector $\vec{w}$ the Perceptron rule will converge to a weight vector (not necessarily unique and not necessarily $\overrightarrow{w^*}$) that gives the correct response for all training patterns and it will do so in a finite number of steps

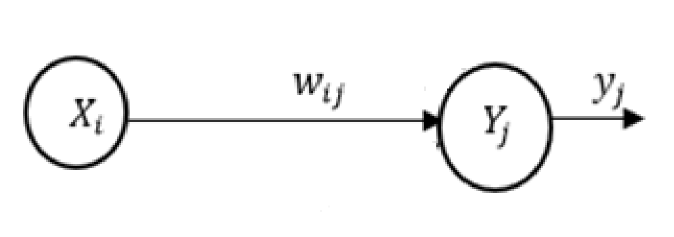
Delta Learning

mansoori@shirazu.ac.ir

Delta Learning Rule

mansoori@shirazu.ac.ir

- Developed by Widrow and Hoff
  - Error-correction rule
  - Least mean squares rule



- The goal is to minimize a cost function based on the error function: $e_j = t_j - y_j$
- Mean square error as cost function: $E = \mathrm{E}\left[\frac{1}{2}\sum_j {e_j}^2\right]$
- Minimizing E with respect to the network parameters (via differentiation) is the method of gradient descent
- How to find the expectation of the process?

Delta Learning Rule

mansoori@shirazu.ac.ir

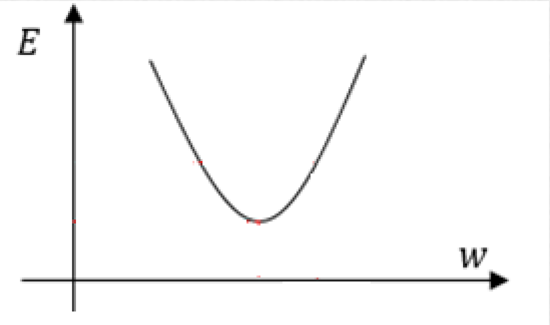
- To avoid its computation, the instantaneous sum of squared errors is used as an approximation of the error function:

$$\xi = \frac{1}{2}\sum_j e_j{}^2 \qquad , \qquad \xi \cong E$$

- Delta learning rule:

$$\Delta w_{ij} = \eta \, e_j \, x_i$$

- A plot of error function and weights is called an error surface

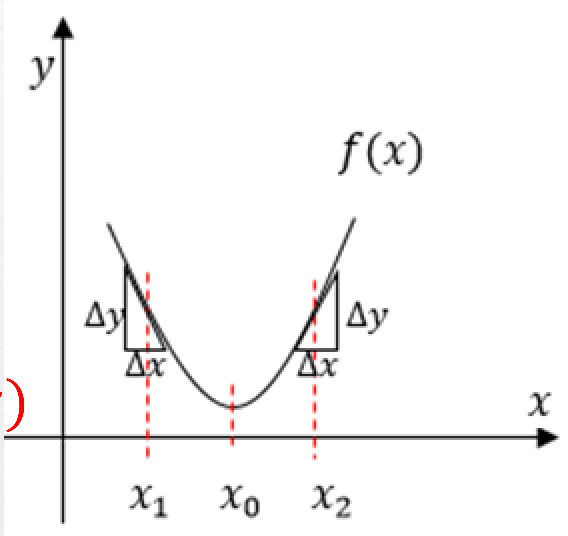


- The minimization process tries to find the minimum point on the surface through an iterative procedure

Delta Rule Derivation

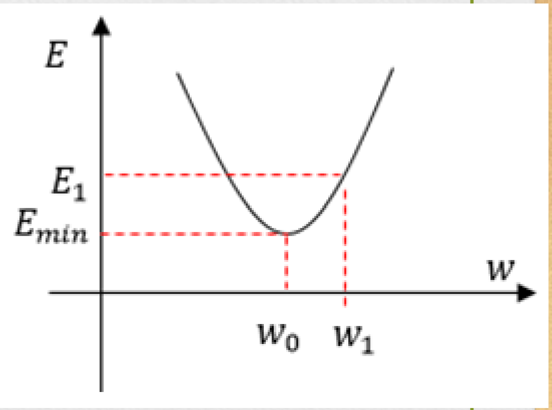
mansoori@shirazu.ac.ir

- $f'(x_1) \approx -\frac{\Delta y}{\Delta x} < 0$ If x_1 decreases: $f'(x_1) \rightarrow -\infty$
- $f'(x_2) \approx \frac{\Delta y}{\Delta x} > 0$: If x_2 increases: $f'(x_2) \rightarrow +\infty$



- So, direction of reaching x_0 is opposite of $f'(x)$ increase

- For 2-class NNs, square error for a training pattern is:
- $E = \frac{1}{2} e^2 = \frac{1}{2} (t - y)^2$



Delta Rule Derivation

mansoori@shirazu.ac.ir

- E is a function of $\vec{w} = [w_1 \; \dots \; w_n]^T$
- Gradient of error: $\nabla E = \frac{\partial E}{\partial \vec{w}} = [\frac{\partial E}{\partial w_1} \; \frac{\partial E}{\partial w_2} \; \dots \; \frac{\partial E}{\partial w_n}]^T$
- Gradient gives direction of most rapid increase in E, so error can be reduced by adjusting w_I in direction of $-\frac{\partial E}{\partial w_I}$

$$\Delta w_I = -\eta \frac{\partial E}{\partial w_I} \quad : \quad \text{gradient descent}$$

$$\frac{\partial E}{\partial w_I} = \frac{\partial}{\partial w_I}\left[\frac{1}{2}(t-y)^2\right] = -(t-y)\frac{\partial}{\partial w_I}y = -(t-y)f'(y\_in)\frac{\partial}{\partial w_I}y\_in$$
$$= -(t-y)f'(y\_in)\frac{\partial}{\partial w_I}(b + \textstyle\sum_{i=1}^{n} x_i w_i\,) = -(t-y)f'(y\_in)\frac{\partial}{\partial w_I}(x_I w_I)$$
$$= -(t-y)f'(y\_in)x_I$$

- If activation function is identity: $y = f(y\_in) = y\_in$, local error can be reduced by: $\Delta w_i = \eta(t-y)\,x_i$

mansoori@shirazu.ac.ir

Competitive Learning

Competitive Learning

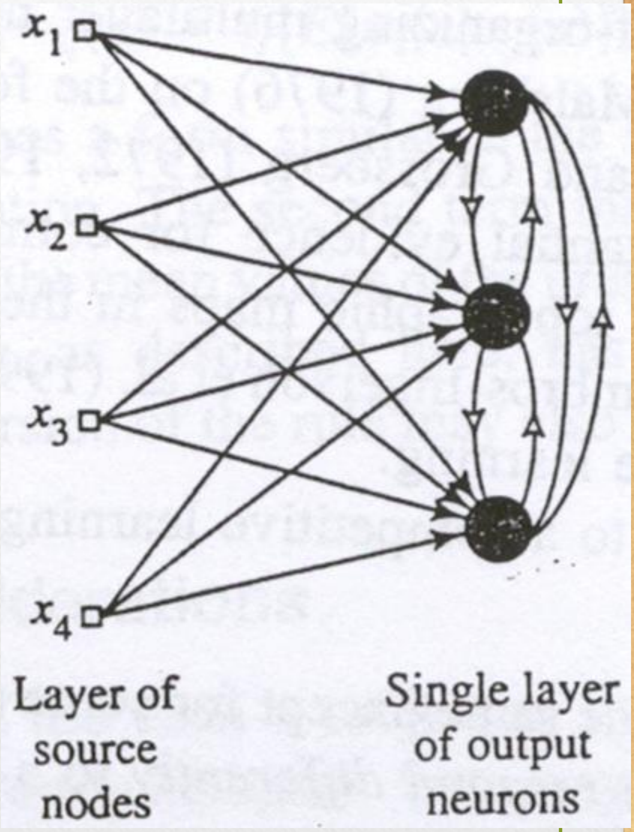
mansoori@shirazu.ac.ir

- Output neurons of an NN (or a group of output neurons) compete among themselves for being active (fired) one
  - At any given time, only one neuron in group is active
  - This behavior naturally leads to identifying features in input data (feature detection)
- Neurobiological basis
  - Competitive behavior was observed and studied in 1970s
- Early self-organizing and topographic map NNs were also proposed in 1970s

Competitive Learning

mansoori@shirazu.ac.ir

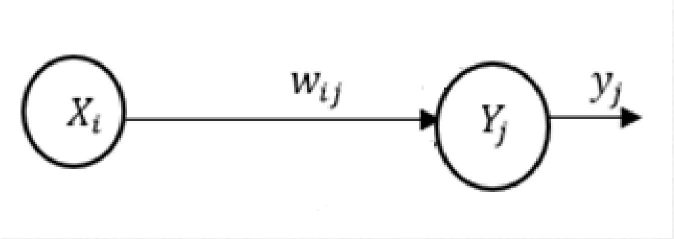
- Elements of competitive learning
  - A set of neurons
  - A limit on the strength of each neuron
  - A mechanism that permits the neurons to compete in respond to a given input, such that only one neuron is active at a time



Competitive Learning

m a n s o o r i @shirazu.ac.ir

- Standard competitive learning rule:



$$\Delta w_{ij} = \begin{cases} \eta\,(x_i - w_{ij}) & \text{, if neuron } Y_j \text{ wins competition} \\ 0 & \text{, otherwise} \end{cases}$$

- Each neuron is allotted a fixed amount of synaptic weight which is distributed among its input nodes:

$$\sum_i w_{ij} = 1 \quad \text{for all } j$$

mansoori@shirazu.ac.ir

Boltzmann Learning

Boltzmann Learning

mansoori@shirazu.ac.ir

- Stochastic learning algorithm based on information-theoretic and thermodynamic principles

- State of network is captured by

  an energy function:

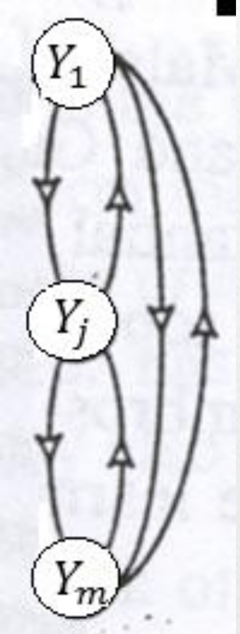
  $$\mathcal{E} = -\frac{1}{2}\sum_i \sum_j w_{ij}\, y_i\, y_j$$

  where

  $y_i \in \{-1,1\}$: bipolar output (state) of neuron Y_i

  and

  $w_{ii} = 0$: none of neurons has self-feedback



Boltzmann Learning

mansoori@shirazu.ac.ir

- Learning process:
  - At each step, choose a neuron at random (say Y_j) and flip its state y_j by following probability:

$$p(y_j \to -y_j) = \frac{1}{1 + e^{-\frac{\Delta \mathcal{E}_j}{T}}}$$

where T is a pseudo-temperature

- State of neurons evolve until thermal equilibrium is achieved

mansoori@shirazu.ac.ir

Supervised/ Reinforced/ Unsupervised Learning

Supervised Learning (SL)

mansoori@shirazu.ac.ir

- SL involves a teacher who has knowledge of the environment and guides the training of the net
- Environment knowledge is in form of input-output examples
  - When viewed as an intelligent agent, this knowledge is current knowledge obtained from sensors
- How SL is applied?
  - Error-correction (delta) learning
- Examples of SL algorithms
  - LMS algorithm
  - Back-propagation algorithm

Reinforcement Learning (RL)

mansoori@shirazu.ac.ir

- RL is supervised learning in which limited information of desired outputs is known
  - Complete knowledge of environment is not available; only basic benefit or reward information
  - A critic rather than a teacher guides learning process
- RL is online learning of an input-output mapping through a process of trial and error, designed to maximize a scalar performance index called reinforcement signal
- RL has roots in experimental studies of animal learning
  - Training a dog by positive (“good dog”, something to eat) and negative (“bad dog”, nothing to eat) reinforcement

Supervised vs. Reinforcement Learning

mansoori@shirazu.ac.ir

Supervised learning	Reinforcement learning
Teacher (detailed information available)	Critic (only reward information available)
Instructive feedback system	Evaluative feedback system
Instantaneous and local information	Delayed and general information
Directed information (how system should adapt)	Undirected information (system has to probe with trial and error)
Faster training	Slower training

Unsupervised Learning (UL)

mansoori@shirazu.ac.ir

- There is no teacher or critic in UL
  - No specific example of the function/model to be learned
- A task-independent measure is used to guide internal representation of knowledge
  - The free parameters of network are optimized with respect to this measure
- Also known as self-organizing when used in the context of NNs
  - NN develops an internal representation of inputs without any specific information
  - Once it is trained, it can identify features in input, based on task-independent criterion

Supervised vs. Unsupervised Learning

mansoori@shirazu.ac.ir

Supervised learning	Unsupervised learning
Teacher (detailed information available)	No specific information available
Instructive feedback system	Task-independent feedback system
Poor scalability	Better scalability

Learning Tasks

mansoori@shirazu.ac.ir

- Pattern classification
- Pattern association
- Data clustering
- Function approximation
- Behavior prediction
- Action control
- System modeling
- Input-output mapping