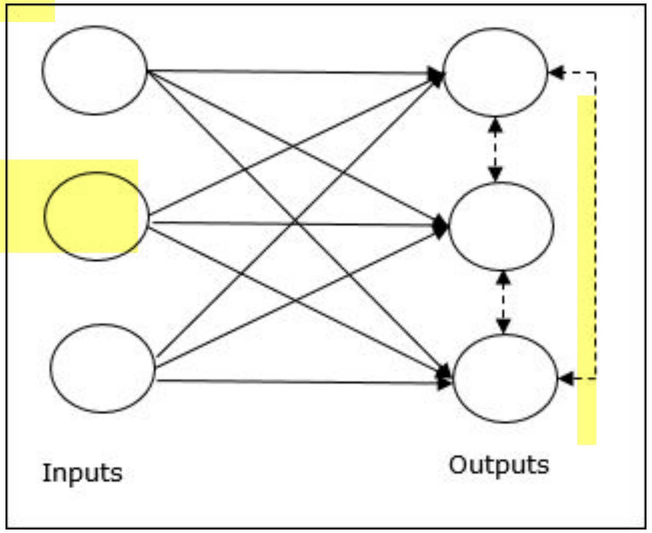
Neural Network

Competitive Learning in NN

Khaleda Akther Papry
Assistant Professor,CSE,DUET
Email: papry.khaleda@duet.ac.bd
Room: 7023 (New academic building)

Competitive Learning in NN

- It is concerned with unsupervised training in which the output nodes try to compete with each other to represent the input pattern. To understand this learning rule we will have to understand competitive net which is explained as follows −
- Basic Concept of Competitive Network
  - This network is just like a single layer feed-forward network having feedback connection between the outputs. The connections between the outputs are inhibitory type, which is shown by dotted lines, which means the competitors never support themselves.



Basic Concept of Competitive Learning Rule

- There would be competition among the output nodes
    - During training, the output unit that has the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.
- Mathematical Formulation

  Following are the three important factors for mathematical formulation of this learning rule −

    - Condition to be a winner: Suppose if a neuron $\boldsymbol{y_k}$ wants to be the winner, then there would be the following condition

$$y_k = \begin{cases} 1 & if\, v_k > v_j\ for\, all\ j,\ j \neq k \\ 0 & otherwise \end{cases}$$

  - It means that if any neuron, say, y_k wants to win, v_k, must be the largest among all the other neurons in the network.

Basic Concept of Competitive Learning Rule

- Condition of the sum total of weight
  - Another constraint over the competitive learning rule is the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron k then

$$\sum_{k} w_{kj} = 1 \quad for\ all\ k$$

- Change of weight for the winner
  - If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows –

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & if\, neuron\, k\, wins \\ 0 & if\, neuron\, k\, losses \end{cases}$$

Here α is the learning rate.

This clearly shows that we are favoring the winning neuron by adjusting its weight and if a neuron is lost, then we need not bother to re-adjust its weight.

K-means Clustering Algorithm

- K-means is one of the most popular clustering based on the concept of partition procedure. We start with an initial partition and repeatedly move patterns from one cluster to another, until we get a satisfactory result.
- Algorithm
  - **Step 1** – Select **k** points as the initial centroids. Initialize **k** prototypes **(w_1,…,w_k)**, for example we can identifying them with randomly chosen input vectors –

$$W_j = i_p, \quad where\, j \in \{1, \ldots, k\}\, and\, p \in \{1, \ldots, n\}$$

  Each cluster **C_j** is associated with prototype **w_j**.

  - **Step 2** – Repeat step 3-5 until E no longer decreases, or the cluster membership no longer changes.

K-means Clustering Algorithm

- **Step 3** − For each input vector $\mathbf{i_p}$ where $\mathbf{p} \in \{\mathbf{1,…,n}\}$, put $\mathbf{i_p}$ in the cluster $\mathbf{C_{j*}}$ with the nearest prototype $\mathbf{w_{j*}}$ having the following relation :

$$|i_p - w_{j*}| \leq |i_p - w_j|, \ j \in \{1, \ldots, k\}$$

- **Step 4** − For each cluster $\mathbf{C_j}$, where $\mathbf{j} \in \{\mathbf{1,…,k}\}$, update the prototype $\mathbf{w_j}$ to be the centroid of all samples currently in $\mathbf{C_j}$, so that

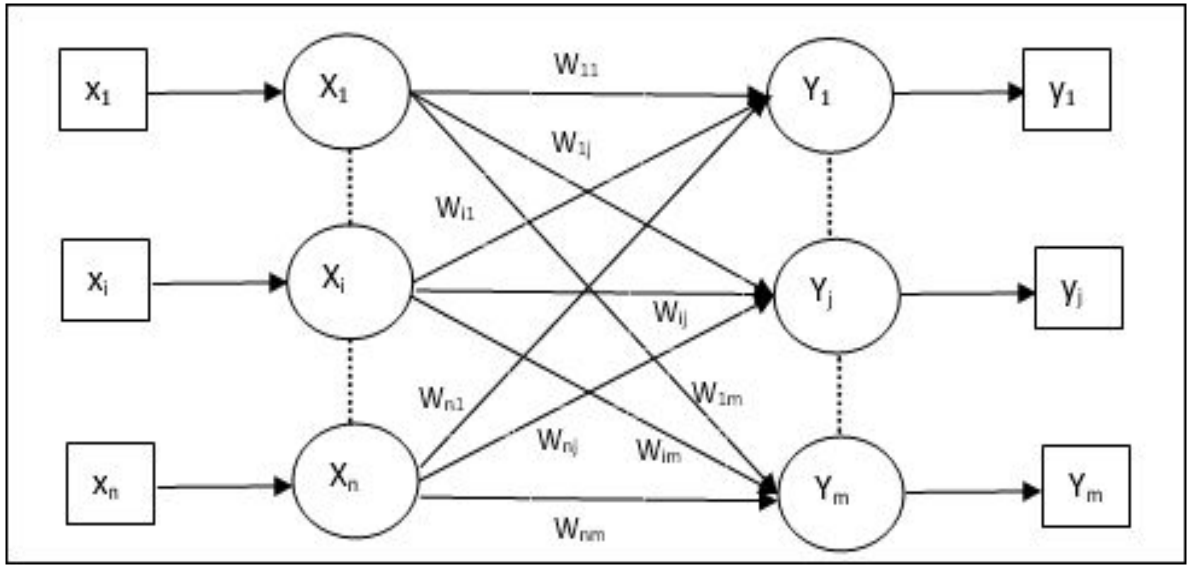
$$w_j = \sum_{i_p \in C_j} \frac{i_p}{|C_j|}$$

- **Step 5** − Compute the total quantization error as follows −

$$E = \sum_{j=1}^{k} \sum_{i_p \in w_j} |i_p - w_j|^2$$

Kohonen Self-Organizing Maps (KSOM)

- Architecture
  - The architecture of KSOM is similar to that of the competitive network.
  - With the help of neighborhood schemes the training can take place over the extended region of the network.



KSOM Algorithm

- Algorithm for training
  - **Step 1** – Initialize the weights, the learning rate **α** and the neighborhood topological scheme.
  - **Step 2** – Continue step 3-9, when the stopping condition is not true.
  - **Step 3** – Continue step 4-6 for every input vector **x**.
  - **Step 4** – Calculate Square of Euclidean Distance for **j = 1 to m**

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

  - **Step 5** – Obtain the winning unit J where Dj is minimum.
  - **Step 6** – Calculate the new weight of the winning unit by the following relation –

$$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

KSOM Algorithm

- **Step 7** – Update the learning rate **α** by the following relation –

$$\alpha(t+1) = 0.5\alpha t$$

- **Step 8** – Reduce the radius of topological scheme.
- **Step 9** – Check for the stopping condition for the network.

- See example from reference book chapter 4.

Learning Vector Quantization (LVQ)

- Architecture :
  - LVQ is quite similar to the architecture of KSOM.
  - **“n”** number of input units and **“m”** number of output units.
  - The layers are fully interconnected with having weights on them.
- Parameters Used

  Following are the parameters used in LVQ training process as well as in the flowchart
  - **x** = training vector $(x_1,\ldots,x_i,\ldots,x_n)$
  - **T** = class for training vector **x**
  - $\mathbf{w_j}$ = weight vector for **j**th output unit
  - $\mathbf{C_j}$ = class associated with the **j**th output unit

LVQ

- Training Algorithm
  - **Step 1** − Initialize reference vectors, which can be done as follows −
    - Step 1a − From the given set of training vectors, take the first “m” number of clusters training vectors and use them as weight vectors. The remaining vectors can be used for training.
    - Step 1b − Assign the initial weight and classification randomly.
    - Step 1c − Apply K-means clustering method.
  - Step 2 − Initialize reference vector α
  - **Step 3** − Continue with steps 4-9, if the condition for stopping criteria is not met.
  - **Step 4** − Follow steps 5-6 for every training input vector **x**.
  - **Step 5** − Calculate Square of Euclidean Distance for **j = 1 to m** and **i = 1 to n**

$$D(j) = \sum_{i=1}^{n}\sum_{j=1}^{m}(x_i - w_{ij})^2$$

LVQ

- **Step 6** − Obtain the winning unit J where Dj is minimum.
- **Step 7** − Calculate the new weight of the winning unit by the following relation −
- **Step 8** − Reduce the learning rate α.
- **Step 9** − Test for the stopping condition. It may be as follows −
    - Maximum number of epochs reached.
    - Learning rate reduced to a negligible value.

LVQ

- Variants
  - Three other variants namely LVQ2, LVQ2.1 and LVQ3
  - Complexity in all these three variants, due to the concept that the winner as well as the runner-up unit will learn, is more than in LVQ
- LVQ2

  The condition of LVQ2 is formed by window. This window will be based on the following parameters −
  - **x** − the current input vector
  - $\mathbf{y_c}$ − the reference vector closest to **x**
  - $\mathbf{y_r}$ − the other reference vector, which is next closest to **x**
  - $\mathbf{d_c}$ − the distance from **x** to $\mathbf{y_c}$
  - $\mathbf{d_r}$ − the distance from **x** to $\mathbf{y_r}$
  - The input vector **x** falls in the window, if $\frac{d_c}{d_r} > 1 - \theta \ and \ \frac{d_r}{d_c} > 1 + \theta$

  Where, θ depends on the number of training samples.

LVQ2

- Updating can be done with the following formula –

$$y_c(t+1) = y_c(t) + \alpha(t)[x(t) - y_c(t)]$$ belongs to different class

$$y_r(t+1) = y_r(t) + \alpha(t)[x(t) - y_r(t)]$$ belongs to same class

  - Here α is the learning rate.
- See examples of LVQ from chapter 4
- Self study: LVQ2.1, LVQ3

THE END