Lecture Series on

Intelligent Control

Lecture 10

Artificial Neural Networks

Competitive Networks

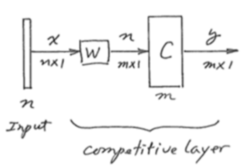
Kwang Y. Lee
Professor of Electrical & Computer Engineering
Baylor University
Waco, TX 76706, USA
Kwang_Y_Lee@baylor.edu

1

Competitive Networks

Competitive Networks

Competitive layer:



$y = \text{compet}(n)$

: find the index i^* of the neuron with the largest net input, and setting its output to 1 (with ties going to the neuron with the lowest index).
All other outputs are set to 0.

$$y_i = \begin{cases} 1, & i = i^* \\ 0, & i \neq i^* \end{cases}, \text{ where } n_{i^*} \geq n_i, \ \forall i, \text{ and } i^* \leq i, \ \forall n_i = n_{i^*}$$

2

Competitive Networks

The prototype vectors are stored in the rows of W.
The net input n calculates the distance between the input vector x and each prototype ${}_iw$ (assuming vectors have normalized lengths of L).
The net input n_i of each neuron i is proportional to the angle θ_i between x and the prototype ${}_iw$:

$$n = Wx = \begin{bmatrix} {}_1w^T \\ {}_2w^T \\ \vdots \\ {}_mw^T \end{bmatrix} x = \begin{bmatrix} {}_1w^T x \\ {}_2w^T x \\ \vdots \\ {}_mw^T x \end{bmatrix} = \begin{bmatrix} L^2 \cos\theta_1 \\ L^2 \cos\theta_2 \\ \vdots \\ L^2 \cos\theta_m \end{bmatrix}$$

The competitive layer assigns an output of 1 to the neuron whose <u>weight vector</u> points in the direction <u>closest</u> to the <u>input vector</u>

3

Competitive Networks

<u>Competitive Learning:</u>

We can now design a competitive network classifier by setting the rows of W to the desired prototype vectors. However, we would like to have a learning rule that could be used to train the weights in a competitive network, without knowing the prototype vectors.

4

Competitive Networks

Recall the instar rule:

$${}_i w(p) = {}_i w(p-1) + \alpha\, y_i(p)\,(x(p) - {}_i w(p-1))$$

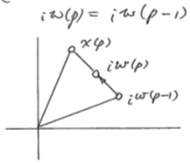
For the competitive network, y is only nonzero for the winning neuron ($i = i^*$). Therefore, we can get the same results using the Kohonen rule.

$${}_i w(p) = {}_i w(p-1) + \alpha\,(x(p) - {}_i w(p-1))$$

$$= (1-\alpha)\,{}_i w(p-1) + \alpha\, x(p)$$

and

$${}_i w(p) = {}_i w(p-1) \qquad i \neq i^*$$

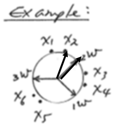


The row of the weight matrix that is closest to the input vector (or has the largest inner product with the input vector) moves toward the input vector.

5

Competitive Networks

<u>Example:</u>



Inputs:

$$x_1 = \begin{bmatrix} -0.1961 \\ 0.9806 \end{bmatrix},\quad x_2 = \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix},\quad x_3 = \begin{bmatrix} 0.9806 \\ 0.1961 \end{bmatrix}$$

$$x_4 = \begin{bmatrix} 0.9806 \\ -0.1961 \end{bmatrix},\quad x_5 = \begin{bmatrix} -0.5812 \\ -0.8137 \end{bmatrix},\quad x_6 = \begin{bmatrix} -0.8137 \\ -0.5812 \end{bmatrix}$$

Initial weights, normalized, random

$${}_1w = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix},\quad {}_2w = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix},\quad {}_3w = \begin{bmatrix} -1.0000 \\ 0.0000 \end{bmatrix},\quad W = \begin{bmatrix} {}_1w^T \\ {}_2w^T \\ {}_3w^T \end{bmatrix}$$

For input x_2:

$$y = \text{compet}(W x_2) = \text{compet}\left(\begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \\ -1.0000 & 0.0000 \end{bmatrix}\begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix}\right)$$

$$= \text{compet}\left(\begin{bmatrix} -0.5547 \\ 0.8321 \\ -0.1961 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The second neuron's weight vector (${}_2w$) was closest to x_2, so it won the competition ($i^* = 2$) and output is 1.

6

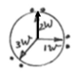
Competitive Networks

Now we apply the Kohonen learning rule to the winning neuron with $\alpha = 0.5$:

$$ {}_2w^{new} = {}_2w^{old} + \alpha (x_2 - {}_2w^{old}) $$

$$ = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} + 0.5 \left(\begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix} - \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \right) = \begin{bmatrix} 0.4516 \\ 0.8438 \end{bmatrix} $$

The Kohonen rule moves ${}_2w$ closer to x_2.

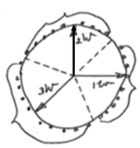


If we continue choosing input vectors at random and presenting them to the network, then at each iteration the weight vector closest to the input vector will move toward that vector.

7

Competitive Networks

Eventually, each weight vector will point at a different cluster of input vectors.
Each weight vector becomes a prototype for a different cluster.



Once the network has learned to cluster the input vectors, it will classify new vectors accordingly.

8

Competitive Networks

<u>Problems with Competitive Layers:</u>

Choice of learning rate: trade-off between the speed of learning and the stability of the final weight vectors.

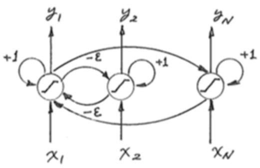
$\alpha \sim 0$: slow learning, once a weight vector reaches the center of cluster, it will tend to stay close to the center.

$\alpha \sim 1$: fast learning, but it will continue to oscillate as different vectors in the cluster are presented.

9

Competitive Networks

Winner-Take-All Networks



The typical competitive neural network consists of a layer of processing elements (PEs), all receiving the same input. The PE with the best output (either maximum or minimum, depending on the criteria) will be declared the winner.

In digital computers, choosing a winner is incredibly simple (just a search for the largest value).

10

Competitive Networks

The concept of choosing a winner, however, often requires a global controller that compares each output with all the others, which is troublesome in distributed systems.

For this and other reasons (e.g., biological plausibility) we wish to construct a network that will find the largest (or smallest) output without global control.

We call this simple system a winner-take-all network.

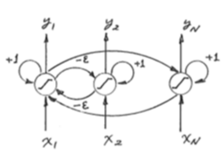
Inputs: $x_1, x_2, \cdots, x_N$

Outputs: $y_1, y_2, \cdots, y_N$

$$y_k = \begin{cases} 1 & x_k \text{ largest} \\ 0 & \text{otherwise} \end{cases}$$

11

Competitive Networks



PE: has a semilinear nonlinearity (clipped linear region), has a self-exciting connection with a fixed weight of +1, laterally connected to all other PEs by negative weights $-\varepsilon$ (lateral inhibition), $0 < \varepsilon < \frac{1}{N}$.

$x_i \geq 0$

Initial condition: Zero output for no input.

Input is presented as an initial condition, i.e., it is presented for one sample and then pulled back.

12

Competitive Networks

The lateral inhibition drives all the outputs toward zero at an exponential rate.

As all the smaller PEs approache zero, the largest PE (PE k) will be less and less affected by the lateral inhibition.

At this time the self-excitation drives its output high, which enforces the zero values of the other PEs.

Hence this solution is stable. Note that the PEs compete for the output activity, and only one wins it.

This network has feeback among the PEs. The output takes sometime to stabilize, unlike the feedforward network, which are instantaneous.

13

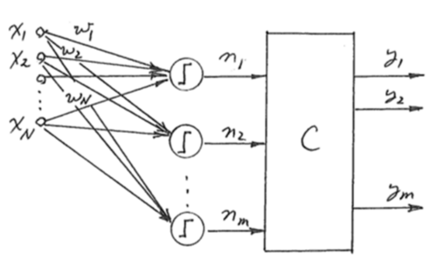
Competitive Networks

Application: The amplitude difference at the input could be small, but at the output it is very clear, so the network amplified the differences, creating a selector mechanism that can be applied to many different applications.

A typical application of the winner-take-all network is at the output of another network (such as a linear associative memory (LAM)) to select the highest output. Thus the net makes a "decision" based on the most probable answer.

14

Competitive Networks



15

Competitive Networks

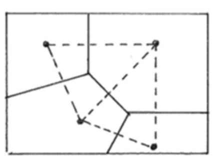
Clustering

The competitive rule allows a single-layer linear network to group and represent data samples that lie in a neighborhood of the input space. Each neighborhood is represented by a single output PE. This operation is commonly called clustering in pattern reconition.

From the point of view of the input space, clustering is dividing the space into local regions, each of which is associated with an output PE. The input space is divided as honeycomb. The weights of each PE represents points in the input space called prototype vectors.

16

Competitive Networks



If we join prototype vectors by a line, its perpendicular bisector will meet other other bisectors, forming a division resembles a honeycomb. Mathematically this division is called Voronoi Tessellation, or simply a tessellation.

Data samples that fall inside the regions are assigned to the corresponding prototype vector. Clustering is therefore a continuous-to-discrete transformation.

17

Competitive Networks

The most important engineering application of competitive learning is vector quantization. In telecommunications, data reduction is needed for economic reasons.

With vector quantization, instead of transmitting the values of each data sample, the data is first categorized in clusters for which the centers, called codebook entries, are known to the transmitter and the receiver. Then just the cluster number is transmitted instead of the data samples.

At the receiver the cluster number is replaced with the codebook entry to recreate the transmitted signal.

18

Competitive Networks

The ultimate requirement of a vector quantizer is to have a set of clusters that minimizes the distance between the centers of each cluster and the input that falls into each cluster.

K-means clustering algorithm: (Duda & Hart 1973)

To find the best division of N samples by K clusters C_i such that the total distance between the clustered samples and their respective centers (i.e., the total variance) is minimized:

$$J = \sum_{i=1}^{K} \sum_{n \in C_i} | x_n - \gamma_i |^2$$

where γ_i is the center of class i.

19

Competitive Networks

$$J = \sum_{i=1}^{K} \sum_{n \in C_i} | x_n - \gamma_i |^2$$

where γ_i is the center of class i.

Steps:
1. randomly assign samples to the class C_i,
2. compute the centers according to
$$\gamma_i = \frac{1}{N_i} \sum_{n \in C_i} x_n$$
3. reassign the samples to the nearest cluster.
4. reiterate

20

Competitive Networks

Gradient estimate:
$$\Delta \gamma_i(n) = \eta \left(x(n) - \gamma_i(n) \right)$$

Recall the competitive rule:
$$w_{i^*}(n+1) = w_{i^*}(n) + \eta \left(x(n) - w_{i^*}(n) \right)$$

where i^* is the PE that wins the competition. All other PEs keep their previous weights.

The gradient estimate is exactly the competitive update if we link the cluster center with the jth PE weight.

Competitive networks thus implement an on-line version of K-means clustering instead of the required batch adaption of K-means.

21

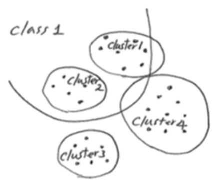
Competitive Networks

Clustering and Classification

Clustering is the process of grouping input samples that are spatial neighbors.

Classification involves the labeling of input samples via some external criterion.

Clustering is an unsupervised process of grouping, Classification is supervised.



Data from each class tends to be dense, and there is a natural valley between classes.
In such cases, clustering can be a preprocessor for classification.

22

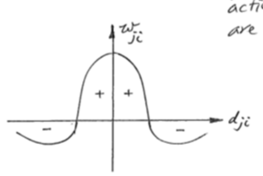
Competitive Networks

Soft Competition:

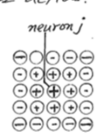
Hard competition: There is only one winner

Soft competition: Not only the winner but also its neighbors are active.

Creates a "bubble" of activity in the output space where the closest PE is the most active (highest output) and its neighbors are less active.



"Maxican Hat" distribution

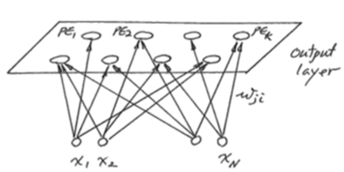


On-Center/Off-Surround

23

Competitive Networks

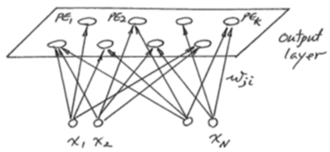
Self-Organizing (Feature) Map



The Kohonen self-organizing map (SOM) network performs a mapping from a continuous input space to a discrete output space, preserving the topological properties of the input. This means that points close to each other in the input space are mapped to the same or neighboring PEs in the output space. The basis of the Kohonen SOM network is soft competition among the PEs in the output space.

24

Competitive Networks



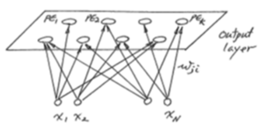
The Kohonen SOM is a fully connected, single-layer linear network. The output is generally organized in a one- or two-dimensional arrangement of PEs, which are called neighborhood.

The SOM network first determines the winning neuron i^*, then the weight vectors for all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule,

$$w_i(n+1) = w_i(n) + \Lambda_{i,i^*}\,\eta(n)\,(x(n) - w_i(n))$$

25

Competitive Networks



$$w_i(n+1) = w_i(n) + \Lambda_{i,i^*}\,\eta(n)\,(x(n) - w_i(n))$$

where Λ_{i,i^*} is a neighborhood function centered at the winning neuron i^*. Typically, both the neighborhood and the step size change with the iteration number.

The neighborhood function Λ is normally a Gaussian:

$$\Lambda_{i,i^*}(n) = \exp\left(\frac{-d^2_{i,i^*}}{2\sigma^2(n)}\right)$$

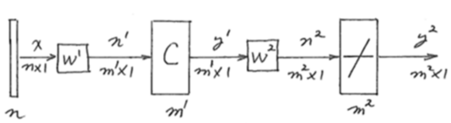
with a variance that decreases with iteration.

26

Competitive Networks

Learning Vector Quantization (LVQ)

: Creating classifiers from competitive networks



$n^1_i = -\|{}_iw^1 - x\|$ $\quad$ $y^2 = W^2 y^1$

$y^1 = \text{compet}(n^1)$

Competitive layer $\quad$ linear layer

Unsupervised $\quad$ Supervised

27

Competitive Networks

In the LVQ network, each neuron in the first layer is assigned to a class, with several neurons often assigned to the same class. Each class is then assigned to one neuron in the second layer.

The winning neuron indicates a subclass, rather than a class. There may be several different neurons (subclasses) that make up each class.

The second layer of the LVQ network is used to combine subclasses into a single class.

$$W^2 = \begin{array}{cl} \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{array}\right] & \longleftarrow \text{class : row } k \\ \uparrow & \\ \text{subclass : column } i & \end{array}$$

$w^2_{ki} = 1$: subclass i is a part of class k

28

Competitive Networks

LVQ Learning: Combines the competitive learning with supervision.

Set of examples (training set):

$\{x_1, t_1\}, \{x_2, t_2\}, \cdots, \{x_n, t_n\}$

Each target vector must contain only zeros, except for a single 1. The row in which the 1 appears indicates the class to which the input vector belongs.

Define W^2: If hidden neuron i is to be assigned to class k, then set $w^2_{ki} = 1$.

29

Competitive Networks

Define W^2: If hidden neuron i is to be assigned to class k, then set $w^2_{ki} = 1$.

Once W^2 is defined, it will never be altered. The hidden weights W^1 are trained with a variation of the Kohonen rule:

At each iteration, an input vector x is presented to the network, and the distance from x to each prototype vector is computed. The hidden neurons compete, neuron i^* wins the competition, and the i^*th element of y' is set to 1.

30

Competitive Networks

Next, y^1 is multiplied by W^2 to get the final output y^2, which also has one nonzero element, k^*, indicating that x is being assigned to class k^*.

The Kohonen rule is used to improve the hidden layer of the LVQ network in two ways.

First, if x is classified correctly, then we move the weights ${}_{i^*}w^1$ of the winning hidden neuron toward x.

$${}_{i^*}w^1(n) = {}_{i^*}w^1(n-1) + \alpha\,(x(n) - {}_{i^*}w^1(n-1)),$$

$$\text{if } y^2_{k^*} = t_{k^*} = 1.$$

31

Competitive Networks

Second, if x was classified incorrectly, then we know that the wrong hidden neuron won the competition, and therefore we move its weights ${}_{i^*}w^1$ away from x.

$${}_{i^*}w^1(n) = {}_{i^*}w^1(n-1) - \alpha\,(x(n) - {}_{i^*}w^1(n-1)),$$

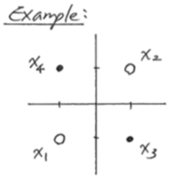
$$\text{if } y^2_{k^*} = 1 \neq t_{k^*} = 0.$$

The result will be that each hidden neuron moves toward vectors that fall into the class for which it forms a subclass and away from vectors that fall into other classes.

32

Competitive Networks

Example:



We want to train an LVQ network to solve the following classification problem:

Class 1: $\left\{ x_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ x_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$

class 2: $\left\{ x_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ x_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\}$

Training sets:

$$\left\{ x_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ t_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\},\ \left\{ x_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ t_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$

$$\left\{ x_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ t_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\},\ \left\{ x_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix},\ t_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

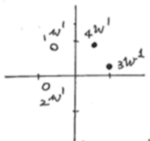
33

Competitive Networks

The output layer weight matrix:

$$W^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

W^2 connects hidden neurons 1 and 2 to output neuron 1,
" 3 and 4 " 2



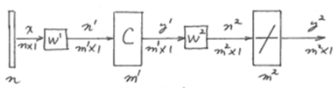
Initial Weights

The hidden layer weight matrix: random

$${}_1w^1 = \begin{bmatrix} -0.543 \\ 0.840 \end{bmatrix},\ {}_2w^1 = \begin{bmatrix} -0.969 \\ -0.249 \end{bmatrix},\ {}_3w^1 = \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix},\ {}_4w^1 = \begin{bmatrix} 0.456 \\ 0.954 \end{bmatrix}$$

34

Competitive Networks



$n_i^1 = -\|{}_iw^1 - x\|$ $\quad$ $y^2 = W^2 y^1$

Present input x_3:

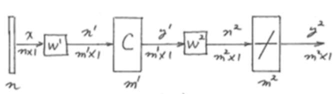
$$y^1 = \text{compet}(n^1) = \text{compet}\left(\begin{bmatrix} -\|{}_1w^1 - x_3\| \\ -\|{}_2w^1 - x_3\| \\ -\|{}_3w^1 - x_3\| \\ -\|{}_4w^1 - x_3\| \end{bmatrix}\right)$$

$$= \text{compet}\left(\begin{bmatrix} -\|[-0.543\ \ 0.840]^T - [1\ \ -1]^T\| \\ -\|[-0.969\ \ -0.249]^T - [1\ \ -1]^T\| \\ -\|[0.997\ \ 0.094]^T - [1\ \ -1]^T\| \\ -\|[0.456\ \ 0.954]^T - [1\ \ -1]^T\| \end{bmatrix}\right) = \text{compet}\left(\begin{bmatrix} -2.40 \\ -2.11 \\ -1.09 \\ -2.03 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The 3rd hidden neuron has the closest weight vector to x_3.

35

Competitive Networks



$n_i^1 = -\|{}_iw^1 - x\|$ $\quad$ $y^2 = W^2 y^1$

The 3rd hidden neuron has the closest weight vector to x_3.
In order to determine which class this neuron belongs to, we multiply y^1 by W^2,

$$y^2 = W^2 y^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

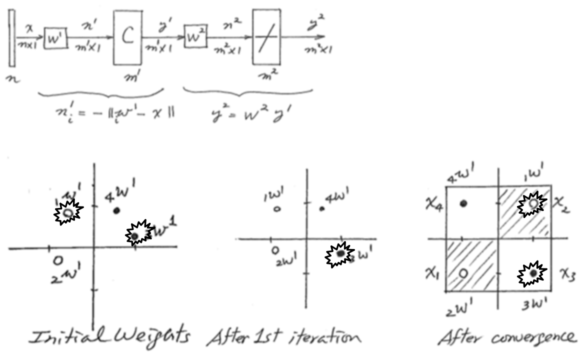
This output indicates that x_3 is a member of class 2.
This is correct, so ${}_3w^1$ is updated by moving it toward x_3,

$${}_3w^1(1) = {}_3w^1(0) + \alpha(x_3 - {}_3w^1(0))$$

$$= \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix}\right) = \begin{bmatrix} 0.998 \\ -0.453 \end{bmatrix}$$

36

Competitive Networks



37