UNSUPERVISED LEARNING: COMPETITIVE LEARNING

INT301 Bio-computation, Week 12, 2021

Clustering Revisit

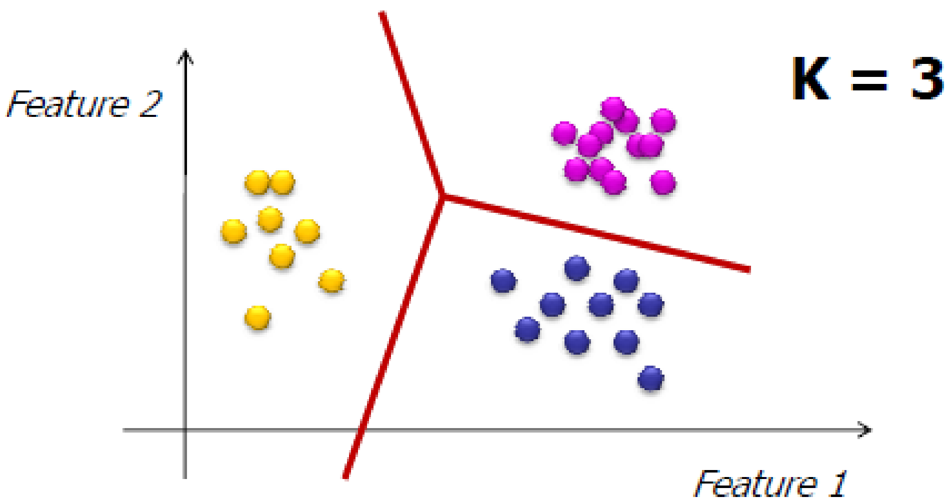
- Clustering analysis is the process of grouping a set of data into clusters
- A cluster is a collection of data points where each observation is
  - Similar to other observations in the same cluster;
  - Dissimilar to observations in other clusters
- Cluster analysis organizes data by abstracting the **underlying structure** either as a grouping of individuals, or as a hierarchy of groups.

Clustering Revisit

- These groupings are based on measured or perceived **similarities** among the patterns.
- Clustering is **unsupervised**. There are no category labels and other information about the source of data.
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms

K-means algorithm Revisit

- The k-means algorithm partitions the data into k *mutually exclusive clusters*

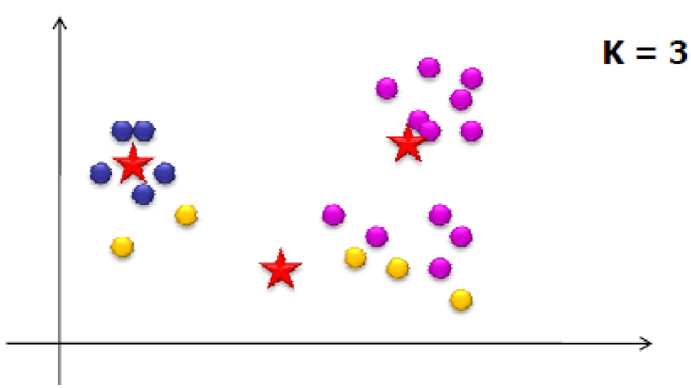


K-means algorithm Revisit

- Objective: minimize the sum of squared distance to its "**representative object**" in each cluster

$$\sum_{i=1}^{K} \sum_{X_j \in S_i} d^2(x_j, \mu_i)$$

S_i is the i-th cluster (i=1,2,…K)
μ_i is the i-th centroid of the points in cluster S_i
d is the distance function

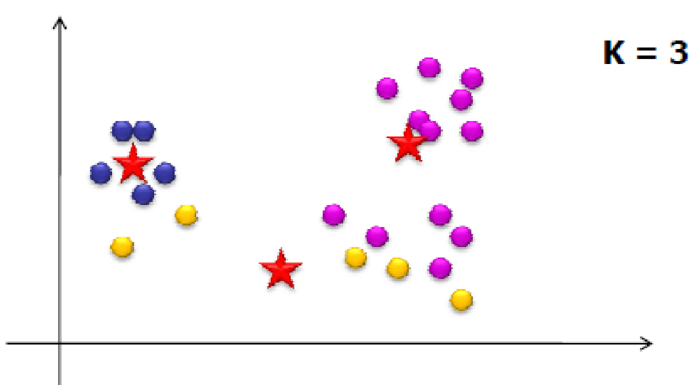


K-means algorithm Revisit

- Basically, the objective is to find the most *compact* partitioning of the data set into k partitions
  - Minimizing intra-cluster variance
  - Maximizing inter-cluster variance
- If we knew the cluster assignment of each point we could easily compute the centroid positions
- If we knew the centroid positions we could easily assign each point to a cluster
- But both are unknown

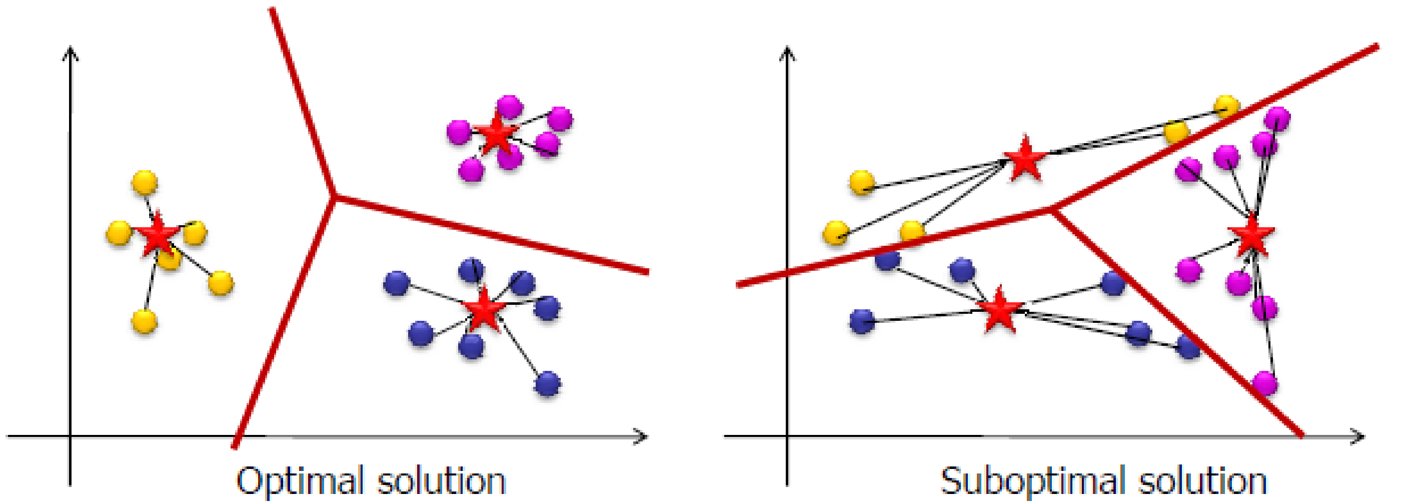
K-means algorithm Revisit

- Choose the number of clusters - K
- Randomly choose initial positions of K centroids
- Assign each of the points to the “nearest centroid” (depends on distance measure)
- Re-compute centroid positions
- If solution converges → Stop!



Things we need to consider

- Does the algorithm guarantee convergence to an optimal solution?

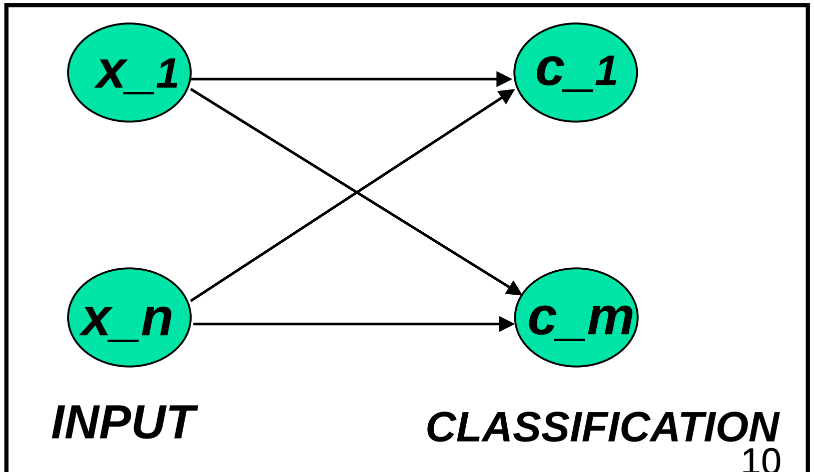


Unsupervised Competitive Learning

- In Hebbian networks, all neurons can “fire” at the same time
- **Competitive learning** means that only a single neuron from each group fires at each time step
- Output units compete with one another.
- These are **winner-takes-all (WTA)** units
- Competition is important for neural networks
  - Competition between neurons has been observed in biological nerve systems
  - Competition is important in solving many problems

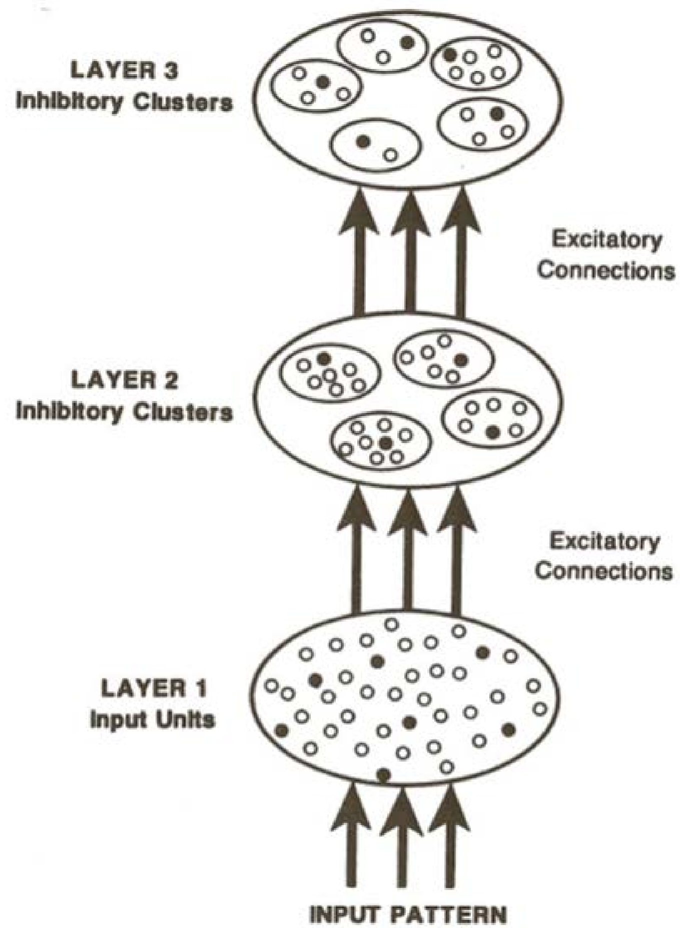
Unsupervised Competitive Learning

- To classify an input pattern into one of the m classes
  - idea case: only one node gives output 1, all the others are 0
  - however, usually more than one nodes have non-zero output
- If these class nodes compete with each other, maybe only one will win eventually and all others lose (winner-takes-all). The winner represents the computed classification of the input



Unsupervised Competitive Learning

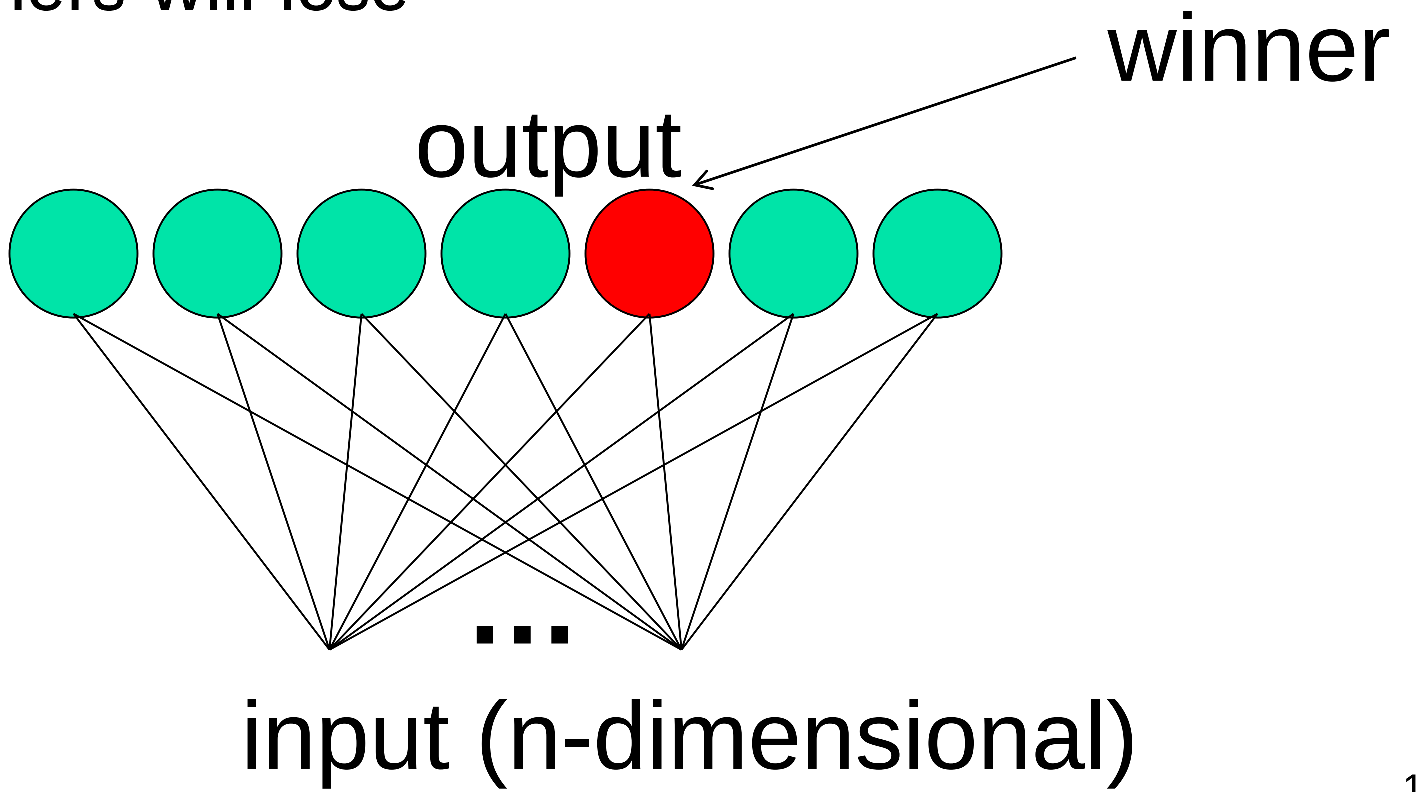
- Units (active or inactive) are represented in the diagram as dots
  - active units are represented by filled dots
  - inactive ones by open dots
- A unit in a given layer can
  - receive inputs from all of the units in the next lower layer
  - project outputs to all of the units in the next higher layer
- Connections between layers are excitatory
- Connections within layers are inhibitory
  - each layer consists of a set of clusters of mutually inhibitory units
  - the units within a cluster inhibit one another in such a way that only one unit per cluster may be active



The architecture of competitive learning mechanism

Winner-takes-all (WTA)

- Among all competing nodes, only one will win and all others will lose

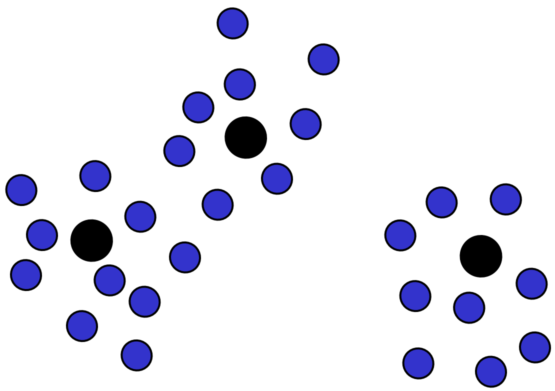


Winner-takes-all (WTA)

- Easiest way to realize WTA: have an external, central arbitrator (a program) to decide the winner by comparing the current outputs of the competitors (break the tie arbitrarily)
- This is biologically unsound (no such external arbitrator exists in biological nerve system)

Simple Competitive Learning

- initialize K prototype vectors
- present a single example
- identify the closest prototype, i.e., the so-called ***winner***
- move the winner even closer towards the example



Intuitively clear, plausible procedure

- places prototypes in areas with high density of data
- identifies the most relevant combinations of features

Simple Competitive Learning

- Winner:

$$h_j = \sum_i w_{ji} x_i$$

$$w_{j*} \cdot x \geq w_j \cdot x \qquad \forall x$$

Note: the inner product of two normal vectors is related to the cosine of the angle between them

Winner = output node whose incoming weights are the shortest Euclidean distance from the input vector

- Lateral inhibition

Simple Competitive Learning

- (One possible) update rule for all neurons:

$$\Delta w_{j*i} = \eta y_j \left(x_i - w_{j*i} \right)$$

$$\begin{cases} y_{j*} = 1 \\ y_j = 0 \quad if \quad j \neq j* \end{cases}$$

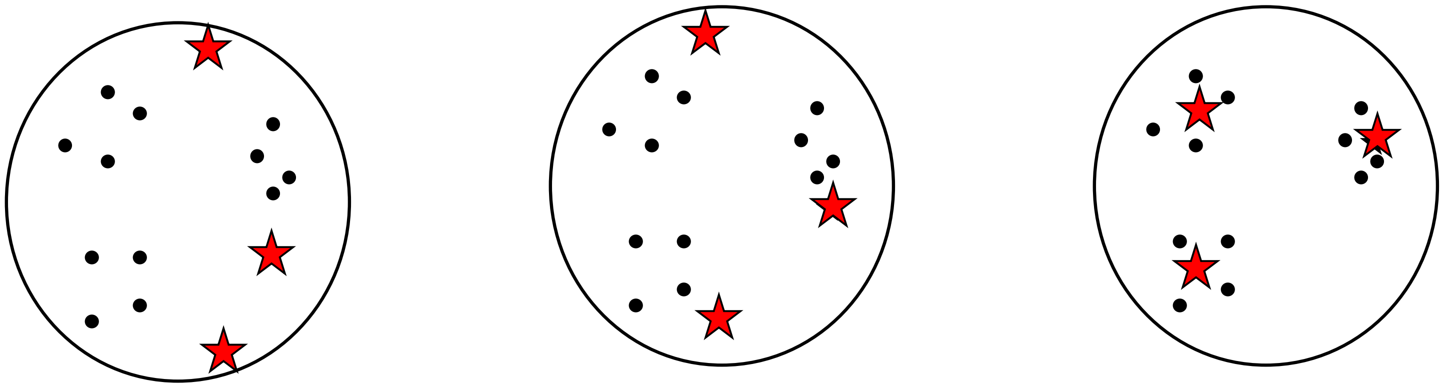
The neuron with largest activation is then adapted to be more likely the input that caused the excitation

Only the incoming weights of the winner node are modified.

Simple Competitive Learning

- Each output unit moves to the center of mass of a cluster of input vectors ➔

clustering



SCL Examples

6 Cases:

(0 1 1) (1 1 0.5)
(0.2 0.2 0.2) (0.5 0.5 0.5)
(0.4 0.6 0.5) (0 0 0)

$$\Delta w_{j^*i} = \eta y_j \left(x_i - w_{j^*i} \right)$$

$$\begin{cases} y_{j^*} = 1 \\ y_j = 0 \quad if \quad j \neq j^* \end{cases}$$

Learning Rate: 0.5

Initial Randomly-Generated Weight Vectors:

[0.14 0.75 0.71]
[0.99 0.51 0.37] Hence, there are 3 classes to be learned
[0.73 0.81 0.87]

Training on Input Vectors

Input vector # 1: [0.00 1.00 1.00]

Winning weight vector # 1: [0.14 0.75 0.71] Distance: 0.41

Updated weight vector:

[0.07 0.87 0.85][0.99 0.51 0.37] [0.73 0.81 0.87]

Input vector # 2: [1.00 1.00 0.50]

Winning weight vector # 3: [0.73 0.81 0.87] Distance: 0.50

Updated weight vector:

[0.07 0.87 0.85][0.99 0.51 0.37]**[0.87 0.90 0.69]**

SCL Examples

$$\Delta w_{j^*i} = \eta y_j (x_i - w_{j^*i})$$

$$\begin{cases} y_{j^*} = 1 \\ y_j = 0 \quad if \quad j \neq j^* \end{cases}$$

```
Input vector # 3:   [ 0.20  0.20  0.20 ]
   Winning weight vector # 2: [ 0.99  0.51  0.37 ] Distance:  0.86
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.59  0.36  0.29 ][ 0.87  0.90  0.69 ]

Input vector # 4:   [ 0.50  0.50  0.50 ]
   Winning weight vector # 2: [ 0.59  0.36  0.29 ] Distance:  0.27
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.55  0.43  0.39 ][ 0.87  0.90  0.69 ]

Input vector # 5:   [ 0.40  0.60  0.50 ]
   Winning weight vector # 2: [ 0.55  0.43  0.39 ] Distance:  0.25
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.47  0.51  0.45 ][ 0.87  0.90  0.69 ]

Input vector # 6:   [ 0.00  0.00  0.00 ]
   Winning weight vector # 2: [ 0.47  0.51  0.45 ] Distance:  0.83
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.24  0.26  0.22 ][ 0.87  0.90  0.69 ]
Finish of Epoch 1
```

SCL Examples

```
Clusters after epoch 1:
 Weight vector # 1: [ 0.07  0.87  0.85 ]
    Input vector # 1:   [ 0.00  1.00  1.00 ]
 Weight vector # 2: [ 0.24  0.26  0.22 ]
    Input vector # 3:   [ 0.20  0.20  0.20 ]
    Input vector # 4:   [ 0.50  0.50  0.50 ]
    Input vector # 5:   [ 0.40  0.60  0.50 ]
    Input vector # 6:   [ 0.00  0.00  0.00 ]
 Weight vector # 3: [ 0.87  0.90  0.69 ]
    Input vector # 2:   [ 1.00  1.00  0.50 ]

Weight Vectors after epoch 2:
  [ 0.03  0.94  0.93 ]
  [ 0.19  0.24  0.21 ]
  [ 0.93  0.95  0.59 ]

Clusters after epoch 2:
   unchanged.
```

SCL Examples

<u>6 Cases</u>

```
(0.9 0.9 0.9)        (0.8 0.9 0.8)
(1 0.9 0.8)          (1 1 1)
(0.9 1 1.1)          (1.1 1 0.7)
```

<u>Other parameters:</u>

```
Three Initial Weight Vectors Generated from Set: {0.8  1.0  1.2}
Learning rate: 0.5
# Epochs: 10
```

- Run same case twice, but with different initial randomly-generated weight vectors.
- The clusters formed are highly sensitive to the initial weight vectors.

SCL Examples

```
Initial Weight Vectors:
  [ 1.20  1.00  1.00 ]
  [ 1.20  1.00  1.20 ]
  [ 1.00  1.00  1.00 ]

Clusters after 10 epochs:
 Weight vector # 1: [ 1.07  0.97  0.73 ]
    Input vector # 3:   [ 1.00  0.90  0.80 ]
    Input vector # 6:   [ 1.10  1.00  0.70 ]
 Weight vector # 2: [ 1.20  1.00  1.20 ]
 Weight vector # 3: [ 0.91  0.98  1.02 ]
    Input vector # 1:   [ 0.90  0.90  0.90 ]
    Input vector # 2:   [ 0.80  0.90  0.80 ]
    Input vector # 4:   [ 1.00  1.00  1.00 ]
    Input vector # 5:   [ 0.90  1.00  1.10 ]

** Weight vector #3 is the big winner & #2 loses completely!!
```

SCL Examples

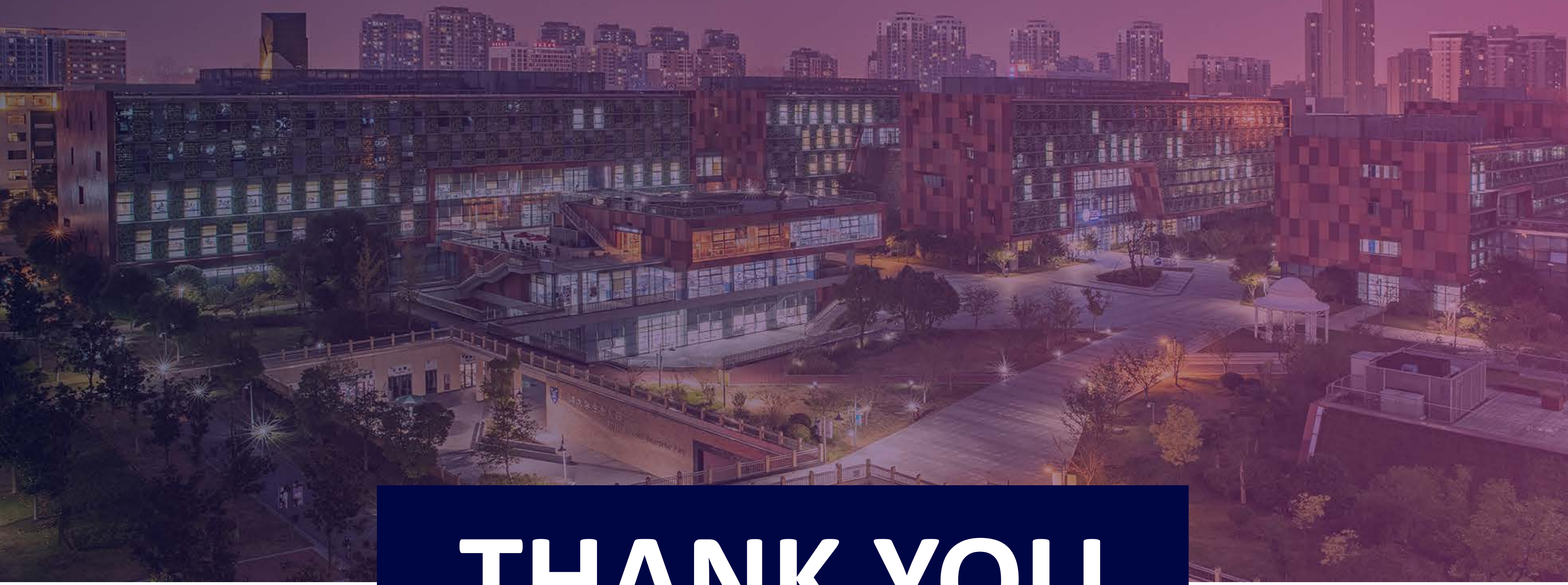
```
Initial Weight Vectors:
  [ 1.00  0.80  1.00 ]   * Better balance of initial weights
  [ 0.80  1.00  1.20 ]
  [ 1.00  1.00  0.80 ]

Clusters after 10 epochs:
 Weight vector # 1: [ 0.83  0.90  0.83 ]
    Input vector # 1:   [ 0.90  0.90  0.90 ]
    Input vector # 2:   [ 0.80  0.90  0.80 ]
 Weight vector # 2: [ 0.93  1.00  1.07 ]
    Input vector # 4:   [ 1.00  1.00  1.00 ]
    Input vector # 5:   [ 0.90  1.00  1.10 ]
 Weight vector # 3: [ 1.07  0.97  0.73 ]
    Input vector # 3:   [ 1.00  0.90  0.80 ]
    Input vector # 6:   [ 1.10  1.00  0.70 ]

 ** 3 clusters of equal size!!
```

Enforcing fairer competition

- Initial position of weight vector of an output unit may be in region with few (if any) patterns

- Some units may never or rarely become a winner, and so weight vector may not be updated, thus preventing it finding richer part of pattern space $\rightarrow$**DEAD UNIT**

- More efficient to ensure a fairer competition where each unit has an equal chance of representing some part of training data



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學