COMP4302/COMP5322, Lecture 6
NEURAL NETWORKS

Clustering.
Self-Organizing Feature Maps (SOM).

Outline

- Clustering
  - K-means algorithm
  - Iterative K-means algorithm
- Competitive Learning
  - Algorithm, Example, Issues
  - Competitive Learning in Biology
- Self-organizing feature maps (SOM)
  - Algorithm
  - Comparison between SOM and K-means
  - Examples
  - Discussion
  - Applications - world poverty map, WEBSom, PicSOM

What is Clustering?

- Clustering – the process of grouping the data into classes (clusters) so that the data objects (examples) are:
  - similar to one another within the same cluster
  - dissimilar to the objects in other clusters
- Clustering is unsupervised classification: no predefined classes
  - Given: A set of unlabeled examples (input vectors) \( p_i \); k – desired number of clusters
  - Task: Cluster (group) the examples into \( k \) clusters
Typical Clustering Applications

- As a stand-alone tool to
  - get insight into data distribution
  - find the characteristics of each cluster
  - assign the cluster of a new example
- As a preprocessing step for other algorithms
  - e.g., dimensionality reduction – using cluster centers to represent data in clusters

Different Ways of Visualizing Clusters

Clustering Example - Stars

- Star clustering based on temperature and brightness (Hertzsprung-Russel diagram)
  - The 3 clusters represent stars in 3 different phases of their life
  - Well-defined clusters
Clustering Example - Animals

- 16 animals described with 13 attributes

<table>
<thead>
<tr>
<th>Animal</th>
<th>small</th>
<th>medium</th>
<th>big</th>
</tr>
</thead>
<tbody>
<tr>
<td>dove</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hen</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>duck</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>goose</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>owl</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hawk</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>eagle</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fox</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>wolf</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>tiger</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lion</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>horse</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>zebra</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cow</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>2 legs</th>
<th>4 legs</th>
<th>hair</th>
<th>hooves</th>
<th>mane</th>
<th>feathers</th>
</tr>
</thead>
<tbody>
<tr>
<td>dove</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hen</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>duck</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>goose</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>owl</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hawk</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eagle</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fox</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wolf</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tiger</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lion</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>horse</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>zebra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cow</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>likes to hunt</th>
<th>run</th>
<th>fly</th>
<th>swim</th>
</tr>
</thead>
<tbody>
<tr>
<td>dove</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>hen</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>duck</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>goose</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>owl</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>hawk</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>eagle</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fox</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>wolf</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tiger</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lion</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>horse</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>zebra</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cow</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Animals Clustering Using Dendrogram

Animals Clustering Using SOM
Clustering Example – Fitting Troops

- Fitting the troops – re-design of uniforms for female soldiers in US army
  - Goal: reduce the number of uniform sizes to be kept in inventory while still providing good fit
- Researchers from Cornell University used clustering and designed a new set of sizes
  - Traditional clothing size system: ordered set of graduated sizes where all dimensions increase together
  - The new system: sizes that fit body types
    - E.g. one size for short-legged, small waisted, women with wide and long torsos, average arms, broad shoulders, and skinny necks

Other Examples of Clustering Applications

- Marketing
  - Help discover distinct groups of customers, and then use this knowledge to develop targeted marketing programs
- Biology
  - Derive plant and animal taxonomies
  - Find genes with similar function
- Land use
  - Identify areas of similar land use in an earth observation database
- Insurance
  - Identify groups of motor insurance policy holders with a high average claim cost
- City-planning
  - Identify groups of houses according to their house type, value, and geographical location

What is a Good Clustering?

- A good clustering method will produce high quality clusters with
  - High intra-class similarity
  - Low inter-class similarity
- The similarity is measured using a distance function
  - E.g. David-Bouldin index – a heuristic measure of the quality of the clustering
    \[ DB = \frac{1}{c} \sum_{i} \max_{j \neq i} \left( \frac{D(x_i) + D(x_j)}{D(x_i, x_j)} \right) \]
    - \( c \) – number of clusters
    - \( D(x_i) \) – mean-squared distance from the points in the cluster \( i \) to the center
    - \( D(x_i, x_j) \) – distance between the centers of cluster \( i \) and \( j \)
  - What is the DB index for a good clustering – big or small?
Distance Measures - Review

- A, B - examples with attribute values \( a_1, a_2, ..., a_n \) & \( b_1, b_2, ..., b_n \)
- Euclidean distance – most frequently used
  \[
  D(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}
  \]
- Manhattan (or city block) distance
  \[
  D(A, B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|
  \]
- Minkowski distance – generalization of both Euclidean and Manhattan
  \[
  D(A, B) = \left( |a_1 - b_1|^q + |a_2 - b_2|^q + \ldots + |a_n - b_n|^q \right)^{1/q}
  \]
  \( q \) – positive integer

K-Means Clustering Algorithm

- Simple and very popular clustering algorithm
- Not a neural network method
- It is an iterative distance-based clustering method
- Requires the number of clusters \( k \) to be specified in advance
- Can be implemented in 4 steps:
  1. Choose \( k \) seeds (vectors with the same dimensionality as the input examples; typically the first \( k \) examples are selected as seeds)
  2. Apply an example, calculate the distance from it to all seeds and assign it to the cluster with the nearest seed point
  3. At the end of each epoch compute the centroid (mean) of the clusters
  4. If the stopping criteria is satisfied (no changes in the assignment of the examples or max number of epochs reached), stop. Otherwise, repeat 2 and 3 with the new centroids taking the role of the seeds.
Iterative K-Means Algorithm

- Each seed has an individual learning rate $\alpha(t) = \frac{1}{t}$
- $t$ – number of input examples for which the seed was the closest one
- The modified algorithm:
  1. Choose $k$ seeds $w_1, w_2, \ldots, w_k$
     Assign individual learning rates $\alpha_1, \ldots, \alpha_k$; set $t_1 = \ldots = t_k = 0$
  2. Apply an example $p_i$, calculate the distance from it to all seeds and assign it to the cluster with the nearest seed point, i.e. the winning seed $w_c$
     Increment $t$ for this seed and update $\alpha_c$: $t_{c}^{\text{new}} = t_{c}^{\text{old}} + 1$, $\alpha_{c}^{\text{new}} = \alpha_{c}^{\text{old}}$
  3. Update the position of the winning seed $w_c$
     $w_c^{\text{new}} = w_c^{\text{old}} + \alpha_c (p_i - w_c^{\text{old}})$
  4. At the end of each epoch check if the stopping criteria is satisfied (no changes in the assignment of the examples or max number of epochs reached). If it is satisfied, stop. Otherwise, repeat 2 and 3.

Iterative K-Means Algorithm - Proof

Show that this algorithms computes the mean of the clusters just like the non-incremental version!

Need for Normalization

- K-means and (clustering algorithms in general) are distance based
- Different attributes are measured on different scales, so the effect of the attributes with smaller scale of measurement will be less significant than those with larger (i.e. some attributes will be considered more important than others)
  $\Rightarrow$ the attribute values must be normalized (standardized)
- e.g. normalization between 0 and 1
  \[
  d_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}
  \]
  $v_i$ – the actual value of attribute $i$
  $\min v_i$ and $\max v_i$ – the minimum and maximum value of $v_i$ taken over all instances in the training set
K-means - Issues

- Different distance measures can be used
  - typically Euclidean distance is used
- Data should be normalized
- Computationally expensive
  - involves finding the distance from each example to each cluster center at each iteration
- Variations
  - Different ways to initialize the seeds
  - Using weights based on how close the example is to the cluster center – Gaussian mixture models
  - Can be used for hierarchical clustering
    - Start with k=2 and repeat recursively within each cluster

Competitive Learning in NNs

- Competitive layers can be used for
  - clustering (e.g. in self-organizing feature map networks - SOM) or
  - classification (e.g. in learning vector quantization networks - LVQ)

Competitive Learning in NNs – Algorithm

- Algorithm:
  1. Choose k prototype vectors \( \mathbf{w}_i, i=1..k \) with the same dimensionality as the input vector
  2. Choose an input vector \( \mathbf{p} \) and calculate the distance between \( \mathbf{p} \) and all prototype vectors (competitive neurons)
  3. Find the index \( i^* \) of the closest competitive neuron and set its output activation to 1 and the activation of all other competitive neurons to 0
  4. Adapt the weight vector of the winner \( \mathbf{w}_i \) using the Kohonen’s rule:
    \[
    \mathbf{w}_i(q+1) = \mathbf{w}_i(q) - \eta \cdot (\mathbf{p}(q) - \mathbf{w}_i(q))
    \]
    \( \eta \) – learning rate, \( \eta < (0,1) \)
  5. Choose another input vector and go to step 2
Competitive Learning in NNs – Algorithm (cont.)

- At the end of the epoch, check if the stopping criteria is satisfied
  - no more changes in the position of the weight vectors (i.e. each weight vector becomes a prototype for a different cluster) or
  - maximum number of epochs is reached

Graphical Representation of Kohonen’s rule

- The prototype vector which is closest to the input vector moves toward the input vector

Example
The First Four Iterations

Typical Convergence (Clustering)

Example

• 6 input vectors:
  \[
  p_1 = \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix},
  p_2 = \begin{bmatrix} 0.9806 \\ -0.1961 \end{bmatrix},
  p_3 = \begin{bmatrix} 0.9806 \\ -0.9806 \end{bmatrix},
  p_4 = \begin{bmatrix} -0.1961 \\ 0.9806 \end{bmatrix},
  p_5 = \begin{bmatrix} -0.9806 \\ -0.9806 \end{bmatrix},
  p_6 = \begin{bmatrix} -0.9806 \\ 0.1961 \end{bmatrix}
  \]

• Let's classify these vectors into 3 clusters — our competitive network will have 3 competitive neurons; random initialization of their weights

\[
  \mathbf{w} = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix},
  \mathbf{w} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix},
  \mathbf{w} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}
  \]

• Present \( p_2 \) calculate the distance to the 3 prototype vectors
  • the distance to \( \mathbf{w} \) is the smallest — set the output of the second competitive neuron to 1 and those of the other competitive neurons to 0
Example – cont.

- Apply Kohonen’s rule to the winning neuron \( i^* = 2 \) \((\alpha = 0.5)\):

  \[
  w^{n+1} = w^{n} + \alpha (p_i - w^{n})
  \]

  \[
  \begin{bmatrix}
  0.7071 \\
  -0.7071 
  \end{bmatrix}
  + \frac{1}{2}
  \begin{bmatrix}
  0.1961 \\
  0.9866
  \end{bmatrix}
  =
  \begin{bmatrix}
  0.7071 \\
  0.4516
  \end{bmatrix}
  \]

  i.e. \( w \) is moved closer to \( p_i \)

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

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

- Once the the network has learned to cluster the input vectors, it can be used to classify new vectors

- Try nnd14c1!

Stability vs. Speed of Learning

- Speed of learning vs. stability of the final weight vectors
  - trade-off forced by the choice of the learning rate \( \alpha \)
  - \( \alpha \) near 0
    - slow learning
    - however, once the weight vector reaches the center of the cluster, it will tend to stay close to the center
  - \( \alpha \) near 1
    - fast learning
    - but once the weight vector has reached a cluster, it will continue to oscillate as different vectors in the cluster are presented

- This trade-off can be used as an advantage!
  - initial training with large \( \alpha \) for fast learning
  - then decrease \( \alpha \) as training progresses to achieve stable prototype vectors

Stability - Example

- If the input vectors don’t fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution
"Dead" Neurons

- Neurons with initial weights far from any input vector may never win, hence never learn \(\Rightarrow\) dead neurons that do nothing
  
  ![Dead Unit](image)

  * will never learn regardless of the order in which vectors are presented

- Solution – add a negative bias to each neuron and decreasing the bias each time the neuron wins; this mechanism is called conscience

Other Disadvantages

- Number of classes is pre-defined – typically 1 neuron per 1 class
  
  - This may not be acceptable for some applications where the number of clusters is not known in advance

- For competitive layers, each class consists of convex region of the input space

  - Competitive layers cannot form classes with non-convex regions or classes that are the union of unconnected regions

  - Convex region (half) – region in which any point can be connected to any other by a straight line that does not cross the boundary of the region

  ![Convex Regions](image)

  - Closed and open convex regions
  - Arbitrary regions

Other Disadvantages – cont.

- Learning is not stable
  
  - Sensitive to weigh vectors initialization; dead neurons

- Each input vector causes a single neuron to fire

  - Not robust to degradation

  - Inconsistent with biological competitive networks
Competitive Layers in Biology

- Neurons are arranged in 2D layers and are interconnected through lateral feedback
- When active, a neuron reinforces not only itself but also those neurons close to it
- A transition from reinforcement to inhibition occurs smoothly as the distance between neurons increases
- This transition can be illustrated using the Mexican-Hat function
  - relates the distance between neurons to the weight of the connections
  - Neurons close to the active neuron have reinforcing (excitatory) connections
  - Neurons beyond a certain distance begin to have inhibitory connections

![](image)

- each neuron $i$ shows the sign and relative strength of its weight $w_{ij}$ going to neuron $j$

Competitive Layers in Biology – cont.

- Biological networks have
  - gradual transition between excitatory and inhibitory regions
  - weaker form of competition than the winner-take-all competition
  - Instead of single active neuron (winner), they have “bubbles” of activity that are centered around the most active neuron

Self-Organising Feature Maps (SOM)

- Extend the competitive learning in several ways
  - Competitive neurons are arranged in some geometry – typically a 2-dimensional grid or a line
  - During learning not only the weights of the winner are updated but also the weights of its neighbors. A neighboring function defines the size and shape of the neighborhood around the winner.
  - As the learning progresses, both the size of the neighborhood and the learning rate gradually decrease
SOM

- 1. Finds the winning neuron $i^*$
- 2. Weights vectors for all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen’s rule:

$$ w_i(q) = w_i(q-1) + \eta(q) \cdot \delta_i(q) $$

$$ \delta_i(q) = \frac{1}{|N_i(d)|} \sum_{j \in N_i(d)} \delta_{ij}(q) $$

$$ \delta_{ij}(q) = \begin{cases} 1 & \text{if } j = i^* \\ 0 & \text{otherwise} \end{cases} $$

$$ N_i(d) = \{ j \mid d_{ij} \leq d \} $$

After many presentations, neighboring neurons will have learned vectors similar to each other

SOM - Algorithm

1. Initialization
   - Select the number of weight vectors; at least 1 competitive neuron corresponds to 1 cluster but the number of neurons is $< \text{the number of training examples}$
   - $w_{ij}(q), 0 \leq i \leq n-1$ – weights from input $i$ to neuron $j$ at time $q$
     Initialize weights to small random values
   - Set the initial radius of the neighborhood to be large

2. Input presentation
   - $p_1(q), p_2(q), \ldots, p_n(q)$, where $p_i(q)$ – input to node $i$ at time $q$

3. Distance calculation
   - $d_i$ – distance between the input and each output neuron $j$:
     $$ d_i = \sum_{j=1}^{n} (p_i(q) - w_{ij}(q))^2 $$

4. Select the winner $j^*$ – output node with minimum distance

5. Weights update for $j^*$ and its neighborhood $N_{j^*}(q)$
   - $w_{ij}(q) = w_{ij}(q) + \eta(q)(p_i(q) - w_{ij}(q)), i \in N_{j^*}(q), 0 \leq i \leq n$
   - $\eta(q)$ - learning rate decreasing in time

6. At the end of each epoch: if stopping criteria is not satisfied (e.g. maximum number of epochs are reached or weights have stopped to move), go to 2.

7. Once training has finished, label SOM
   - each training example is applied and the winning competitive neuron is labeled with it
   - allows to find which examples are grouped together
   - will activate the same winner or neighboring winners
   - To find the cluster of a new example, apply it and find the winning competitive neuron
SOM – Grid and Neighborhood

- Neurons in an SOM do not have to be arranged in a 2D grid
- 1D, 3D or more dimensional arrangements can be used
- Distance can be defined in different ways
- Not necessary Euclidean distance
- Different neighborhoods can be used
  - rectangular, hexagonal, circular
  - The convergence of the network is not sensitive to the exact shape of the neighborhoods


SOM – practical algorithm

1. Initialization of the weights
   - \( w_{ij}(t), 0 \leq i \leq n-1 \) – weights from input \( i \) to neuron \( j \) at time \( t \)
   - Initialized weights to small random values

2. Initialization of the neighborhood kernel \( h_r(t) \); \( s \) – the winner
   - circular, \( r \) is over all nodes in the grid
   - rectangular, \( r \) is over the neighbors of the winner \( s \)
   - \( \eta(t) & \sigma(t) \) - monotonically decreasing functions of time (their exact forms are not critical; thus can be selected linear)

Meaning
- learning rate decreases for stability
- the width of the neighborhood kernel shrinks after each iteration until it includes only the winner

SOM – practical algorithm 2

3. Input vector presentation
   - \( x_1(t), x_2(t), \ldots, x_n(t) \), where \( x_i(t) \) – input to node \( i \) at time \( t \)

4. Distance calculation
   - \( d_j \) – distance between the input and each output neuron
     \[
     d_j^2 = \sum (x(t) - w_j(t))^2
     \]

5. Select the winner \( j^* \) output node with minimum distance

6. Weights update for \( j^* \) and its neighborhood
   - \( u_{ij}(t+1) = u_{ij}(t) + h_r(t)[x(t) - w_j(t)] \)

7. If stopping criteria is not satisfied (e.g. maximum number of epochs are reached or weight vectors stop to move), go to 3.

7. Label SOM
SOM Dynamics – Example 1
- 2 dimensional input examples, 4 well defined clusters
- 4 competitive neurons arranged in a 2D grid (2 x 2)

- after 100 iterations
- after 5000 iterations

SOM Dynamics – Example 2
- 2 dimensional input examples
- 25 competitive neurons arranged in a 2D grid (5 x 5)

- Initial weight vectors
- Input vectors

- each 2 element vector is represented by a dot
- dots of neighboring neurons are connected

- randomly selected from this region and presented to the network

SOM Example - Convergence
Discussion

- Each time a vector is presented, the neuron with the closest weight vector will win the competition.
- The winning neuron and its neighbors move their weight vectors closer to the input vector (and therefore to each other).
- Two tendencies of the weight vectors:
  - Spread out over the input space as more input examples are presented.
  - Move toward the weight vectors of neighborhood neurons.
- These 2 tendencies work together to rearrange the neurons in the layer so that they represent the input space according to its distribution.
  - Evenly in this example as the input vectors were generated with equal probability from any point in the input space => neurons classify roughly equal areas of the input space.
- Try nn14fm2 – 2D SOM!

1D SOM - Example

Try nn14fm1 – 1D SOM!

From Haykin, Neural Networks – a Comprehensive Foundation.

SOM Example – Animals Data

- 16 examples (animals) described with 13 attributes
- 3 experiments:
  a) 2 competitive neurons arranged in a 2D grid (1 x 2)
  b) 9 competitive neurons arranged in a 3 x 3 grid
  c) 15 competitive neurons arranged in a 5 x 5 grid
- After convergence the competitive neurons were labeled
  - each example applied and the winning neuron labeled with the animal’s name.
SOM Example – Animals Data (cont)

- How many clusters?
- SOM creates clusters but finding their boundaries is difficult
  - Requires background knowledge
  - There are some algorithms that can help
- The characteristics of the data in the clusters also cannot be told from the map
  - Requires analysis of the data

Comparison between K-means and SOM

- Both of them
  - Define a mapping from a high dimensional space (n-number of examples) to a lower dimensional space (k-number of clusters)
  - Use more units to represent regions of the input space with higher density
  - Require the number of clusters to be specified in advance
- K-means is a **hard** competitive learning algorithm (winner-take-all algorithm) - Each input example determines the adaptation of 1 unit (the winner)
- SOM is a **soft** competitive learning algorithm – not only the winner but also some other units are adapted (the neighborhood of the winner)
- K-means does not learn topology, SOM does:
  - Similar inputs are mapped to neighboring neurons
  - Topologically close units in SOM have similar input vectors mapped on them

Some Applications of SOM

- World Poverty Map
  - Data: 39 indicators describing various quality of life factors, e.g. state of health, nutrition, education services, etc. (data from World Bank, 1992, for ~150 countries)
WEBSOM

- WEBSOM - http://websom.hut.fi/websom/
  - Organizes text documents onto meaningful maps for interactive browsing and search
  - Based on SOM: related documents appear close to each other like the books on the shelves of a well-organized library
  - Different data used: e.g. 7 million patent abstracts (largest document collection used), Usenet newsgroup articles, etc.
  - User may zoom view any area of the map by clicking
WEBSOM - View Levels

- Whole map
- Zoomed map
- Map node
- Single document

WEBSOM - Method

- SOMs are used at 2 levels
  - To form word category map representing each document
    - Takes into consideration semantic similarities of words
  - To organize the documents into a document map
    - Uses histogram of the word occurrence from the word map

PicSOM

- Content-based image retrieval – given a set of reference images
  - PicSOM retrieves similar images
    - Try it at http://www.cis.hut.fi/picsom/ftp.sunet.se
  - Database used in the demo: Swedish FTP images (4350 images)
  - SOM is used to organize images into 2D map so that similar images are located near each other
  - A hierarchical SOM called Tree Structured Self-Organizing Map (TS-SOM) is used
    - Similarities can be based on color, texture and shape (separate feature vectors are formed for each of these)
    - A distinct TS-SOM is constructed for each feature vector set and these maps are used in parallel to select the returned images
    - Queries are performed through a web interface and the queries are iteratively refined as the system exposes more images to the user
Some SOM Links

- NN research Lab in Helsinki
  http://www.cis.hut.fi/
- Bibliography on SOM and LVQ
  http://liinwww.ira.uka.de/bibliography/Neural/SOM.LVQ.html
- SOM tutorial – Java applets
  http://davis.wpi.edu/~matt/courses/soms

Summary and Conclusions

- Competitive networks learn
  - To cluster (categorize) the input vectors (define a mapping from a high to a lower dimensional space)
  - the distribution of inputs by allocating more neurons to classify parts of the input space with higher densities of input
- SOMs are competitive networks
  - they also learn the topology of the input vectors – neurons next to each other in the network respond to similar inputs
  - in contrast to the basic competitive networks, in SOM not only the winner is adapted but also its neighborhood
- Clustering algorithms
  - are distance based and require no normalization of the input data
  - do not have built-in feature selection mechanism (more variables are not necessary better); a good idea is to apply a dimensionality reduction algorithm (e.g. PCA) to access the importance of the attributes before clustering