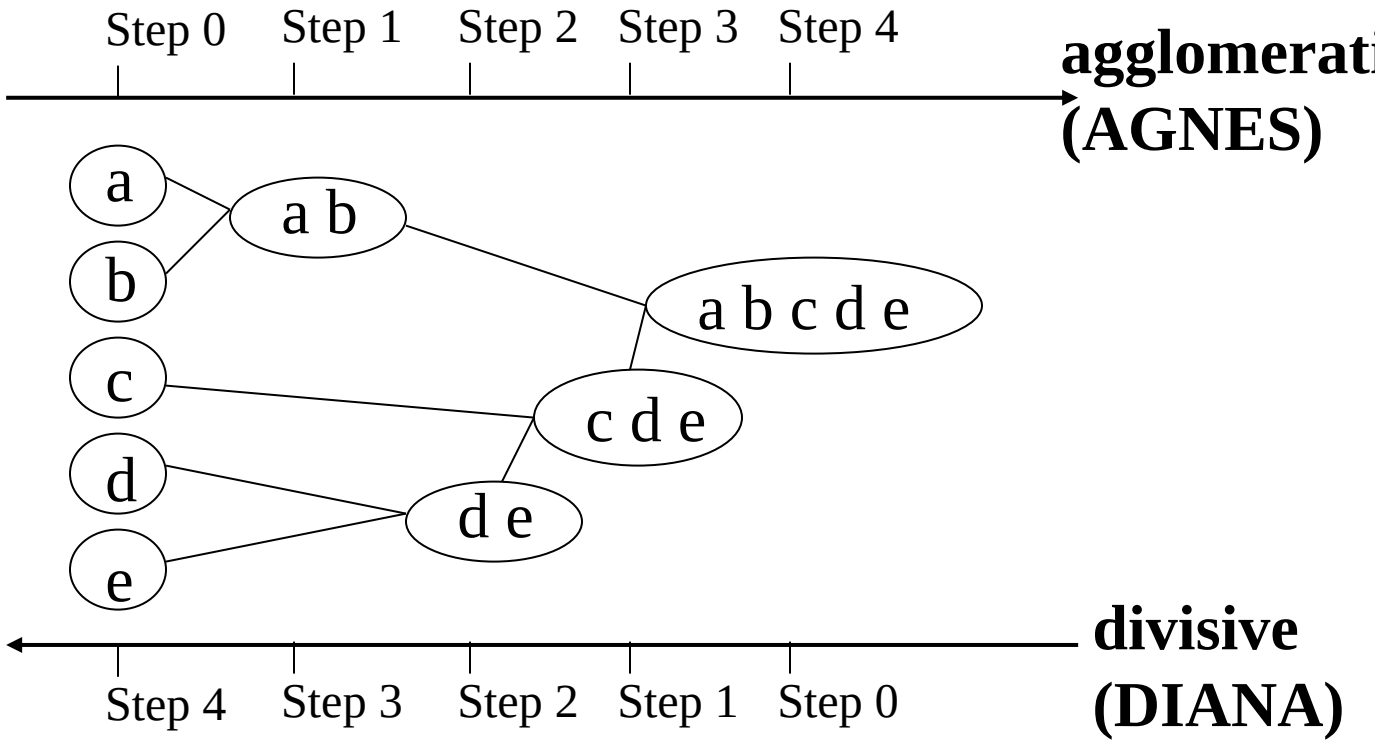
Hierarchical Clustering

- Use distance matrix as clustering criteria
- This method does not require the number of clusters ***k*** as an input, but needs a termination condition



BIRCH

- Incrementally construct a **CF (Clustering Feature) tree**, a hierarchical data structure for multiphase clustering
  - Phase 1: scan DB to build an **initial in-memory CF tree** (a multi-level compression of the data that tries to preserve the inherent clustering structure of the data)
  - Phase 2: use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree
- *Scales linearly*: finds a good clustering with a single scan and improves the quality with a few additional scans
- *Weakness:* handles only numeric data, and sensitive to the order of the data record

BIRCH

Clustering Feature (CF): *CF = (N, LS, SS)*

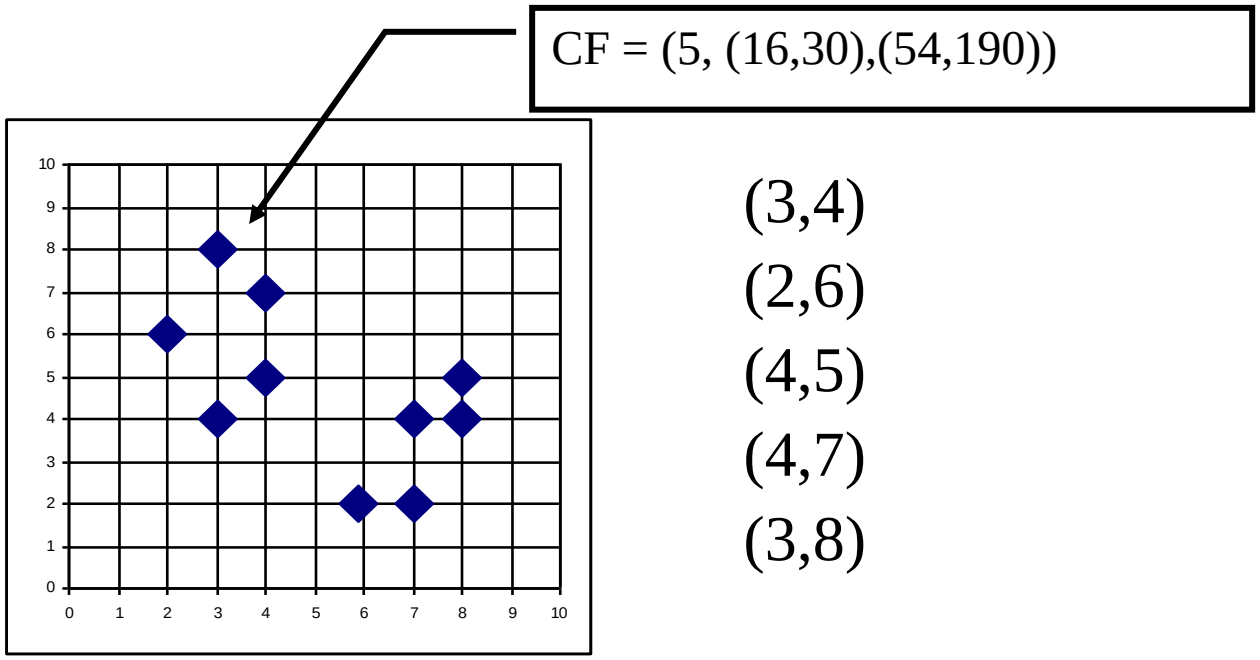
N: Number of data points

LS: linear sum of N points:

$$\sum_{i=1}^{N} X_i$$

SS: square sum of N points

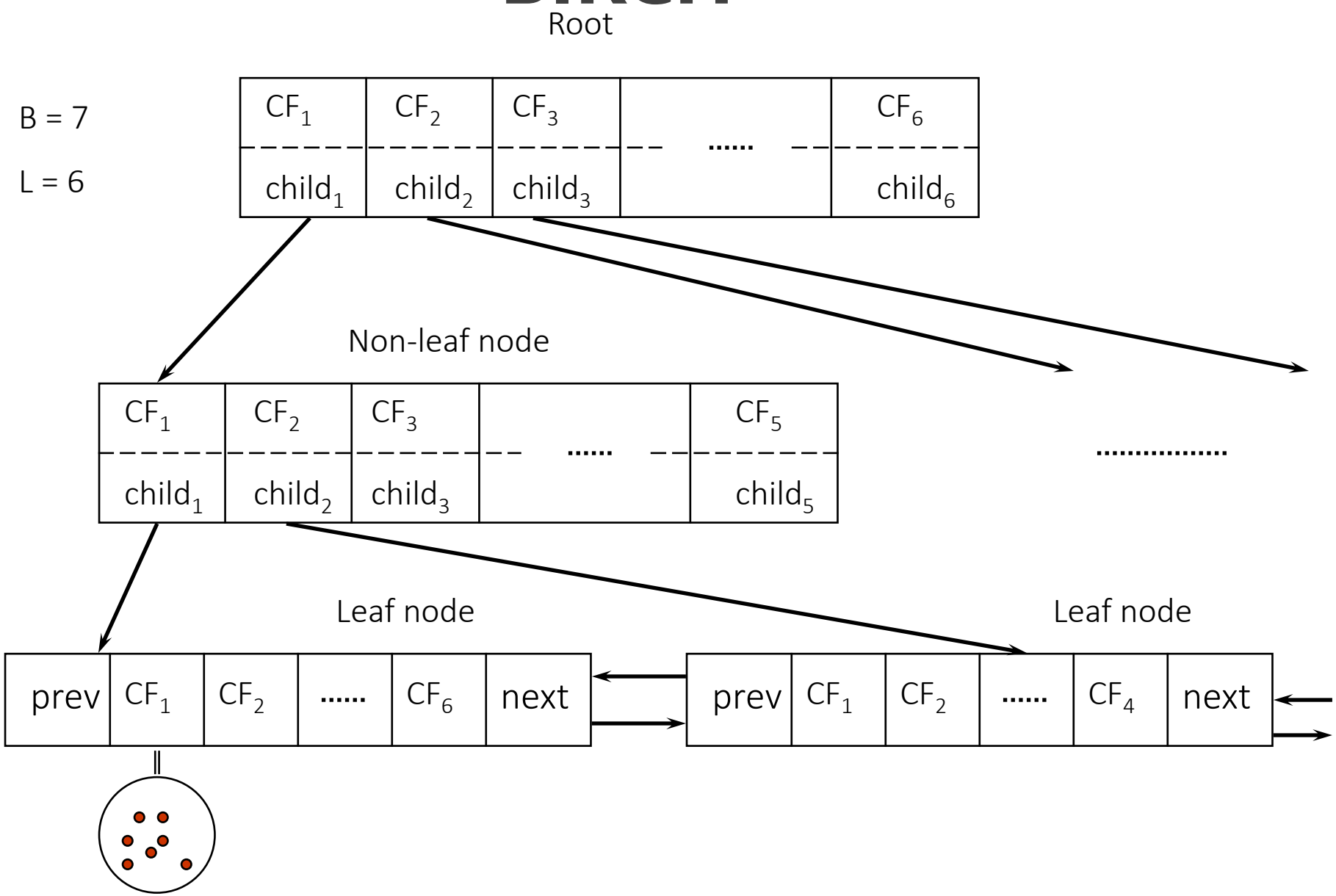
$$\sum_{i=1}^{N} X_i^2$$



BIRCH

- Clustering feature:
  - Summary of the statistics for a given subcluster: the 0-th, 1st, and 2nd moments of the subcluster from the statistical point of view
  - Registers crucial measurements for computing cluster and utilizes storage efficiently
- A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering
  - A nonleaf node in a tree has descendants or “children”
  - The nonleaf nodes store sums of the CFs of their children
- A CF tree has two parameters
  - **Branching factor B: max # of children**
  - **Threshold L: max diameter of sub-clusters stored at the leaf nodes**

BIRCH


Root
B = 7
L = 6
CF1
CF2
CF3
.....
CF6
child1
child2
child3
child6
Non-leaf node
CF1
CF2
CF3
.....
CF5
child1
child2
child3
child5
.....
Leaf node
Leaf node
prev
CF1
CF2
.....
CF6
next
prev
CF1
CF2
.....
CF4
next

BIRCH

- Cluster Diameter

$$\sqrt{\frac{1}{n(n-1)}\sum(x_i - x_j)^2}$$

- For each point in the input
  - Find closest leaf entry
  - Add point to leaf entry and update CF
  - If **entry diameter > max_diameter**, then split leaf, and possibly parents
- Algorithm is O(n)
- Concerns
  - Sensitive to insertion order of data points
  - Since we fix the size of leaf nodes, so clusters may not be so natural
  - Clusters tend to be spherical given the radius and diameter measures

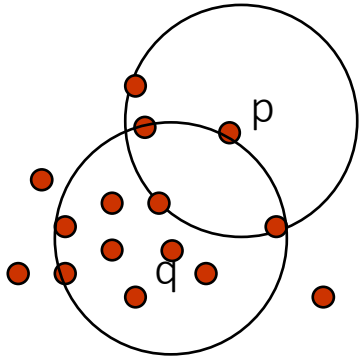
Density Based Algorithms

- Clustering based on density (local cluster criterion), such as density-connected points
- Major features:
  - Discover clusters of arbitrary shape
  - Handle noise
  - One scan
  - Need density parameters as termination condition
- Several interesting studies:
  - DBSCAN: Ester, et al. (KDD'96)
  - OPTICS: Ankerst, et al (SIGMOD'99).
  - DENCLUE: Hinneburg & D. Keim (KDD'98)
  - CLIQUE: Agrawal, et al. (SIGMOD'98) (more grid-based)

Density Based Algorithms

- Two parameters*:*
  - *Eps*: Maximum radius of the neighbourhood
  - *MinPts*: Minimum number of points in an Eps-neighbourhood of that point
- $N_{Eps}(p)$: {q belongs to D | dist(p,q) ≤ Eps}
- Directly density-reachable: A point *p* is directly density-reachable from a point *q* w.r.t. *Eps*, *MinPts* if
  - *p* belongs to $N_{Eps}(q)$
  - core point condition:

    $|N_{Eps}(q)| \geq MinPts$

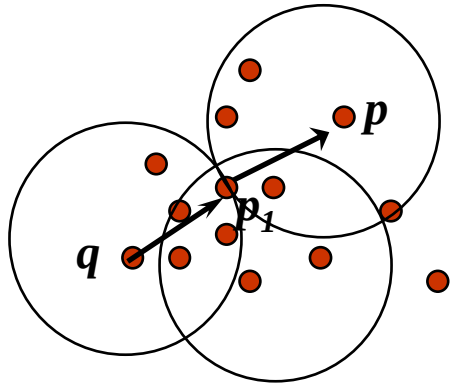


MinPts = 5

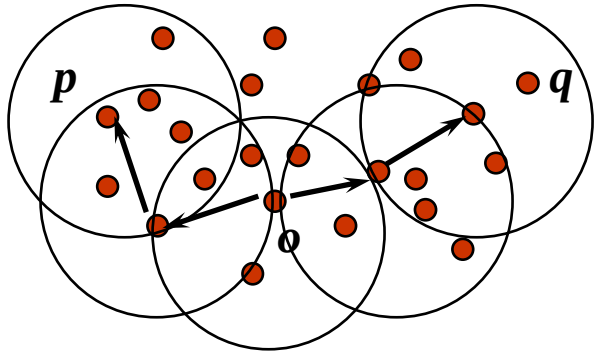
Eps = 1 cm

Density Based Algorithms

- **Density-reachable:**
  - A point *p* is density-reachable from a point *q* w.r.t. *Eps*, *MinPts* if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i

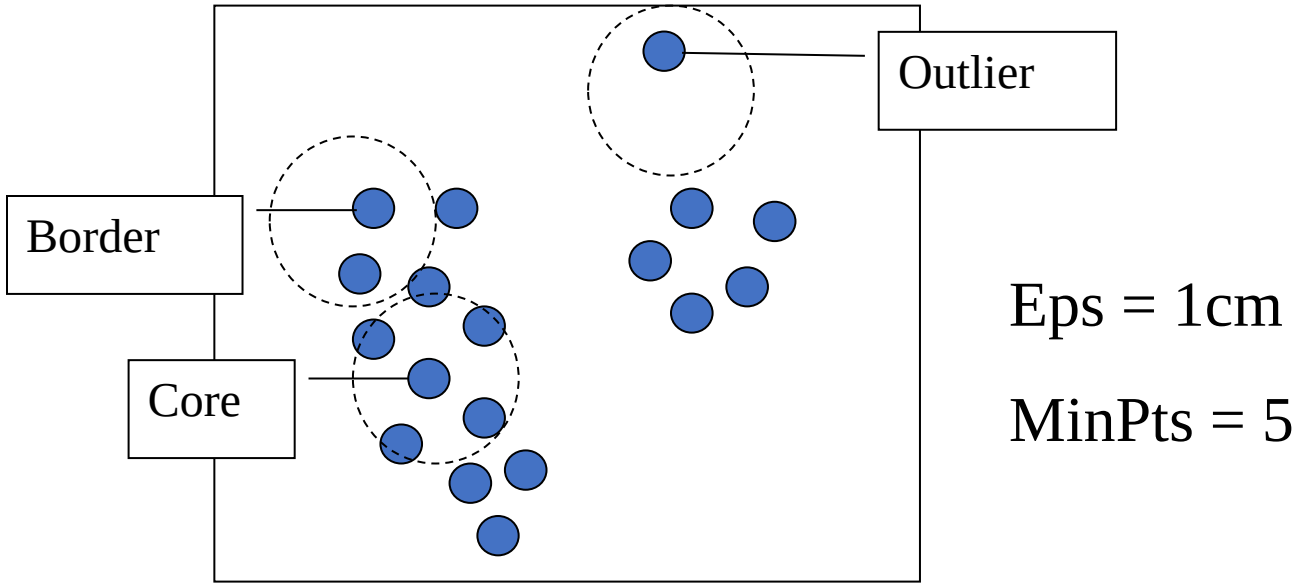


- **Density-connected**
  - A point *p* is density-connected to a point *q* w.r.t. *Eps*, *MinPts* if there is a point *o* such that both, *p* and *q* are density-reachable from *o* w.r.t. *Eps* and *MinPts*



DBSCAN

- Relies on a *density-based* notion of cluster: A *cluster* is defined as a maximal set of density-connected points
- Discovers clusters of arbitrary shape in spatial databases with noise



DBSCAN

- Arbitrary select a point *p*
- Retrieve all points density-reachable from *p* w.r.t. *Eps* and *MinPts*
- If *p* is a core point, a cluster is formed
- If *p* is a border point, no points are density-reachable from *p* and DBSCAN visits the next point of the database
- Continue the process until all of the points have been processed

DBSCAN

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

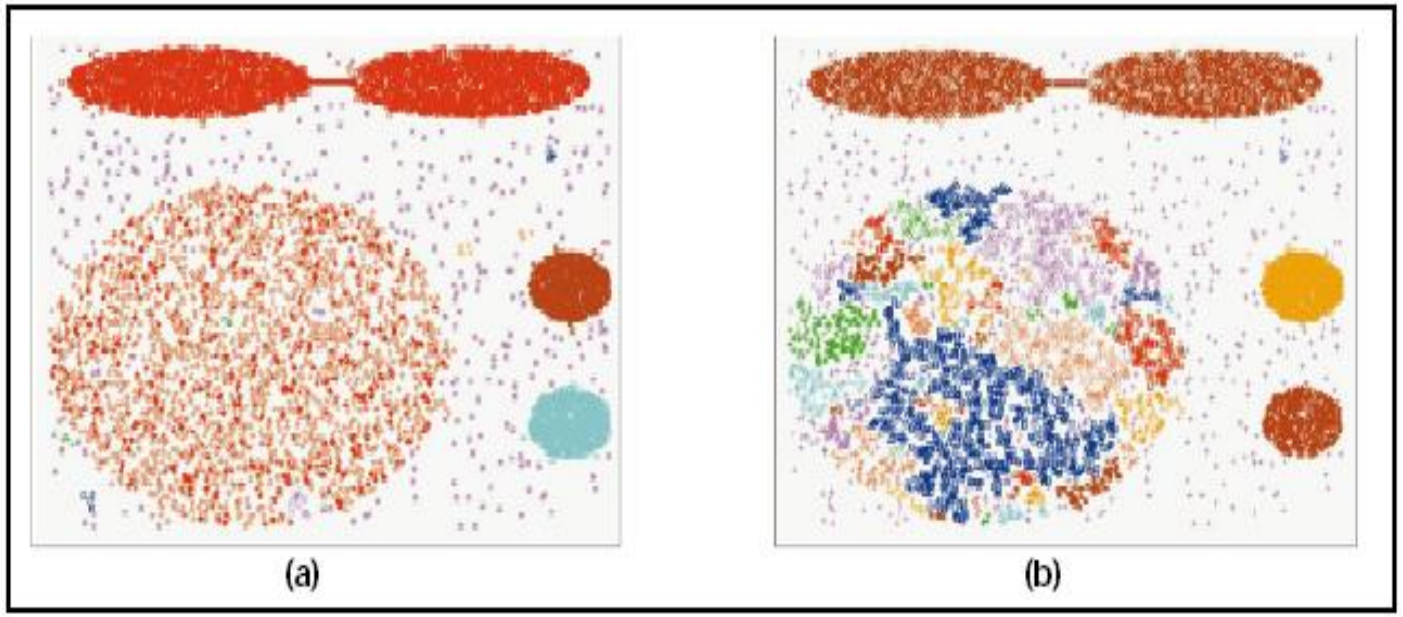
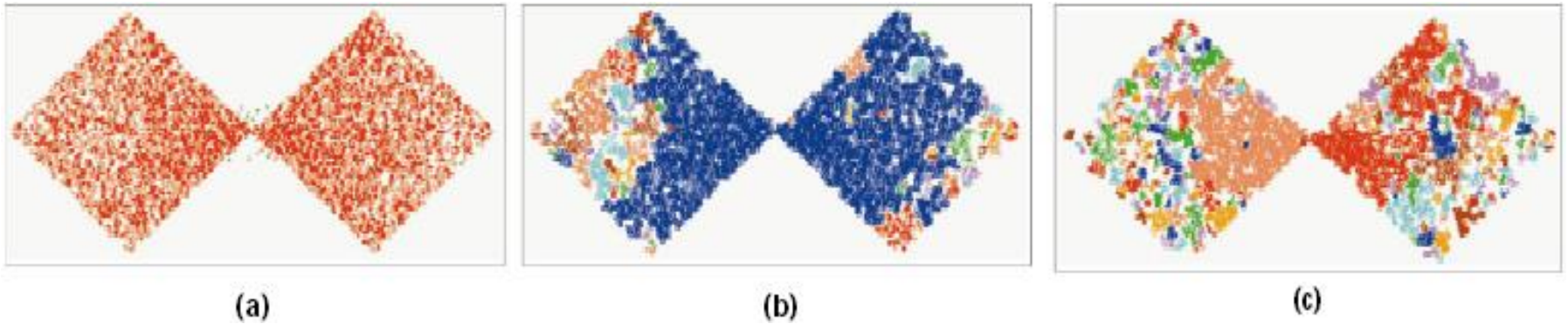


Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.



OPTICS

- Index-based:

  k = number of dimensions

  N = 20

  p = 75%

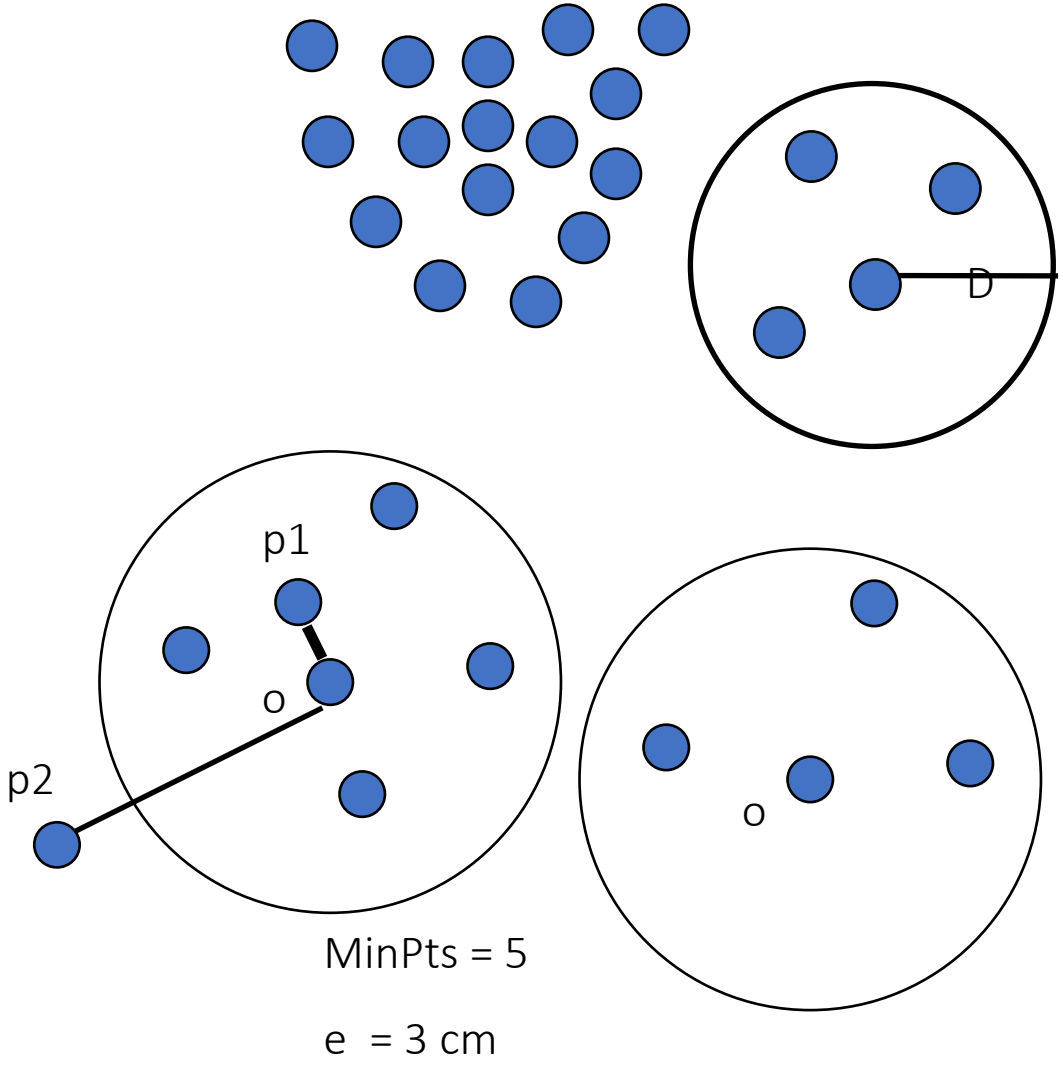
  M = N(1-p) = 5

  - Complexity: O(*NlogN*)
- Core Distance:

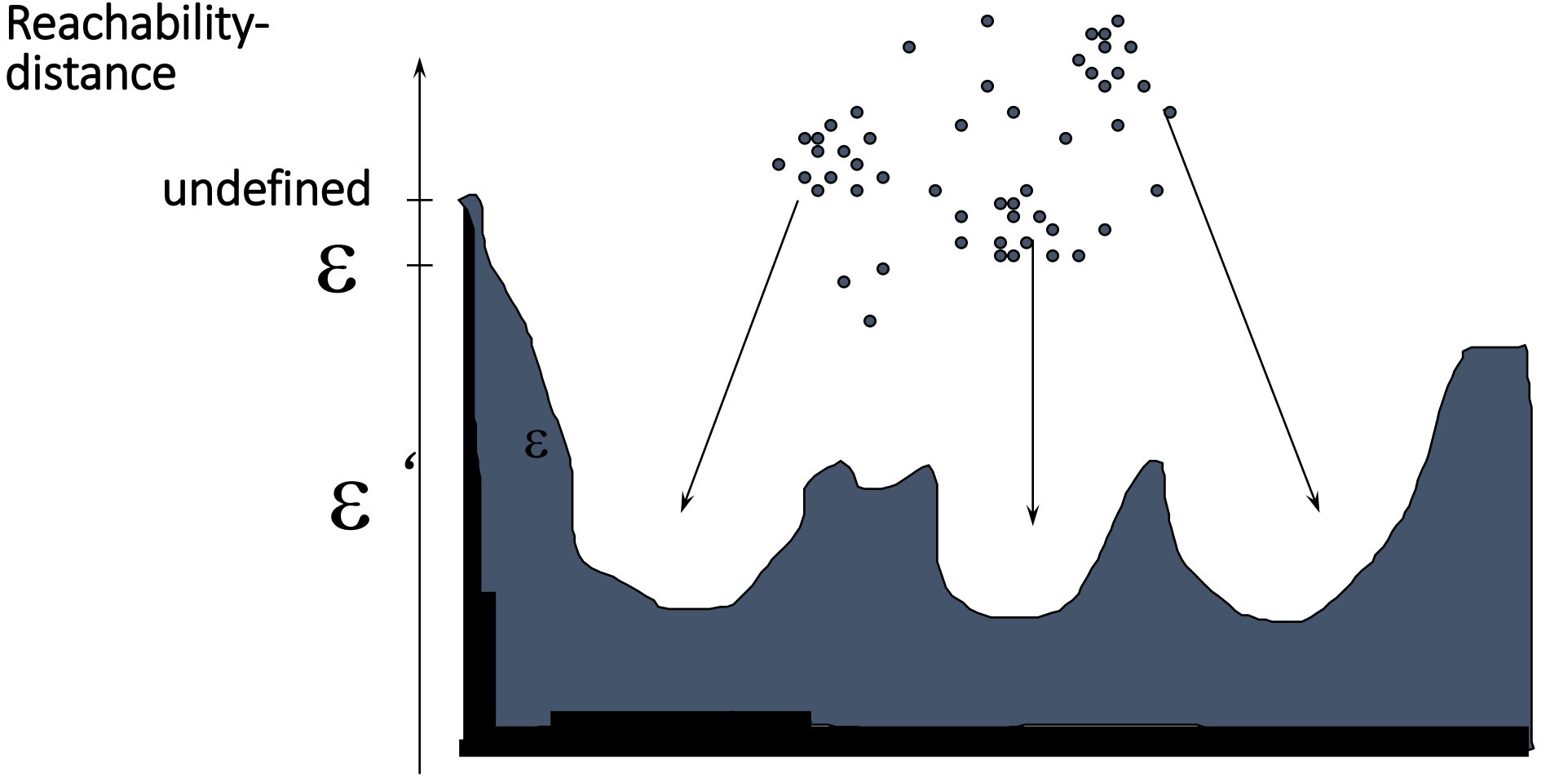
  min eps s.t. point is core
- Reachability Distance

Max (core-distance (o), d (o, p))

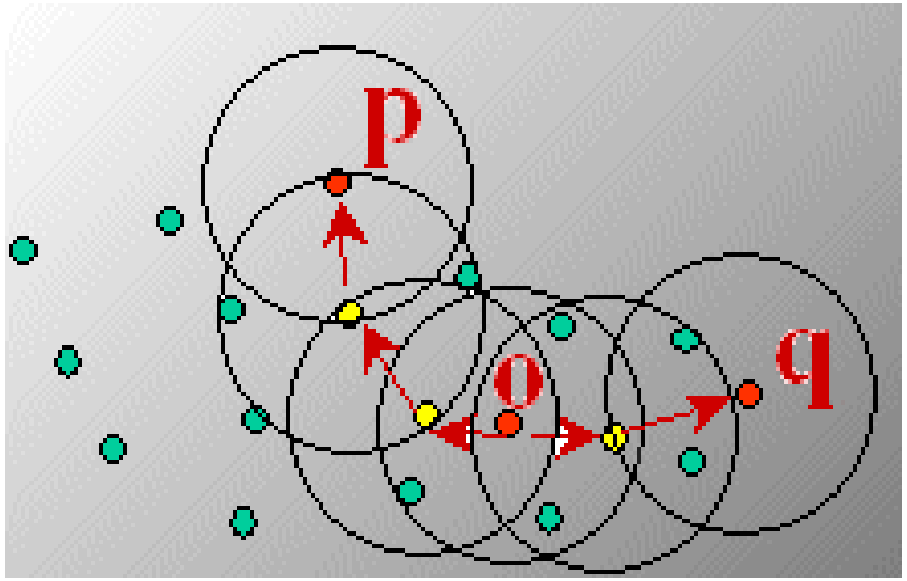
r(p1, o) = 2.8cm. r(p2,o) = 4cm

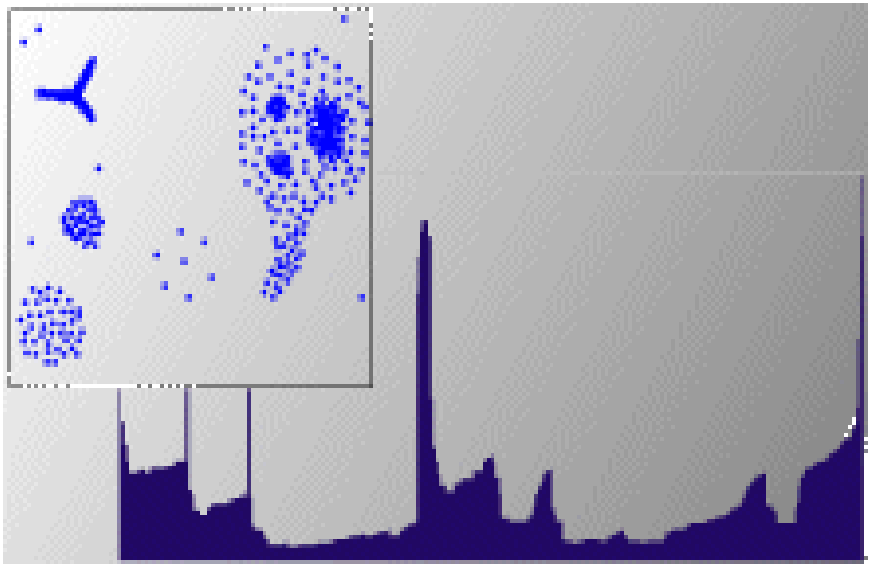


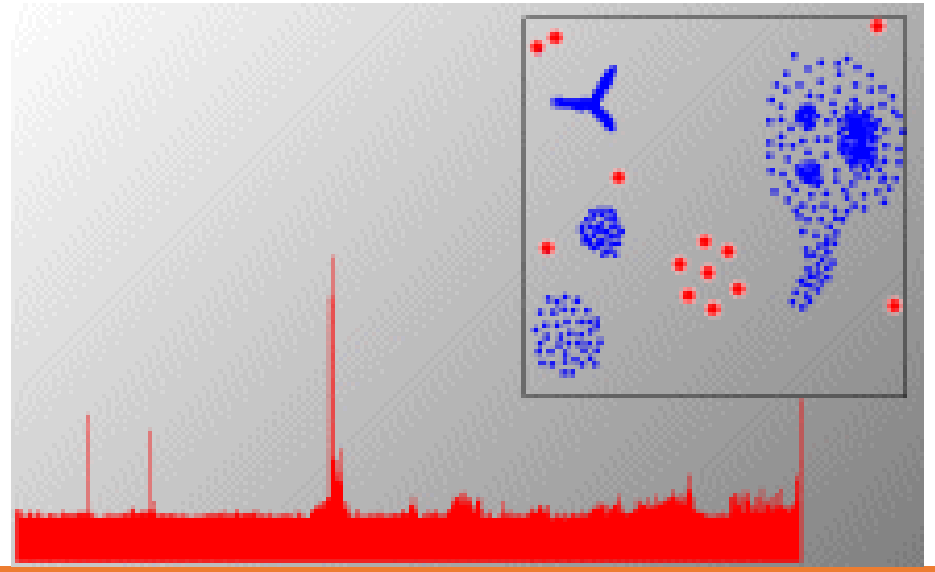
OPTICS

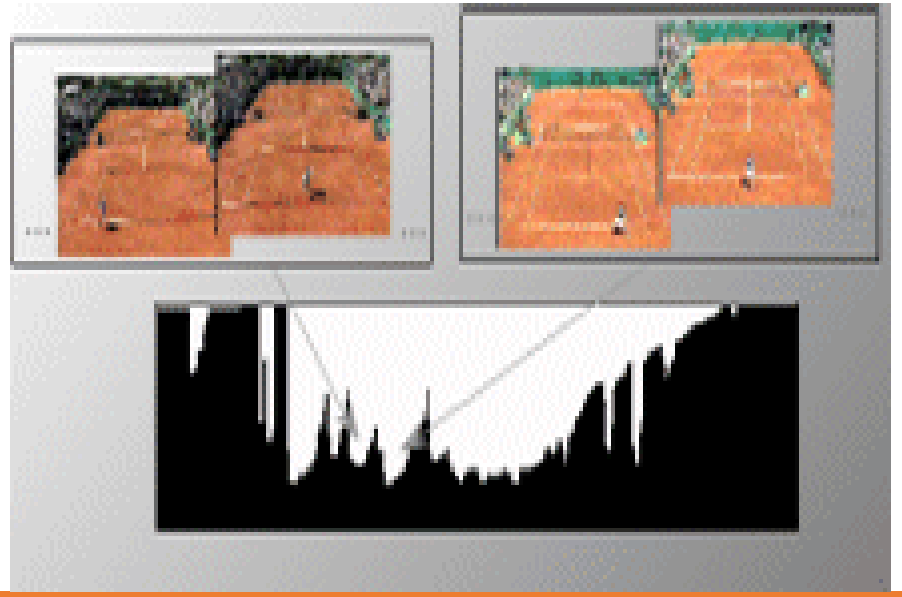

Reachability-distance
undefined
ε
ε'
ε
Cluster-order of the objects

OPTICS







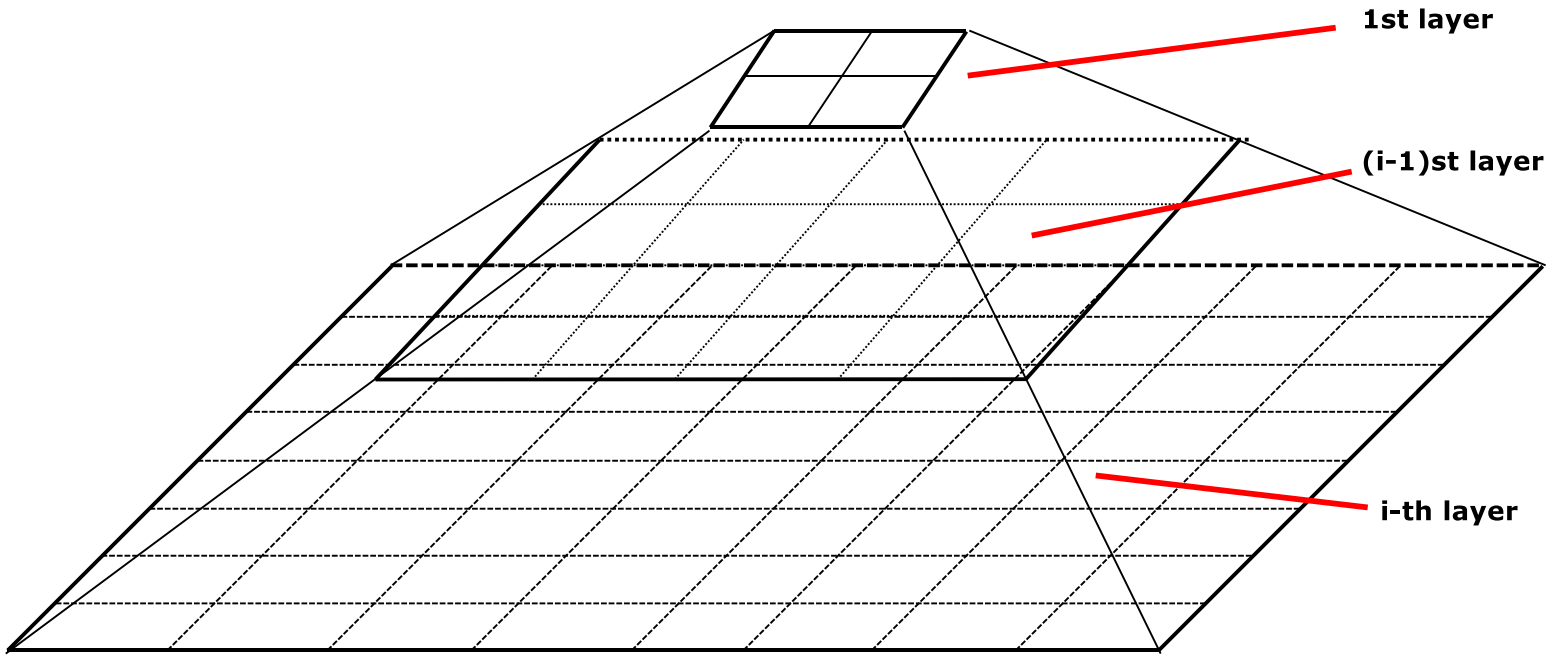


Grid Based Algorithms

- Using multi-resolution grid data structure
- Several interesting methods
  - STING (a STatistical INformation Grid approach) by Wang, Yang and Muntz (1997)
  - WaveCluster by Sheikholeslami, Chatterjee, and Zhang (VLDB'98)
    - A multi-resolution clustering approach using wavelet method
  - CLIQUE: Agrawal, et al. (SIGMOD'98)
    - Both grid-based and subspace clustering

STING

- Wang, Yang and Muntz (VLDB'97)
- The spatial area is divided into rectangular cells
- There are several levels of cells corresponding to different levels of resolution



STING

- Each cell at a high level is partitioned into a number of smaller cells in the next lower level
- Statistical info of each cell is calculated and stored beforehand and is used to answer queries
- Parameters of higher level cells can be easily calculated from parameters of lower level cell
  - count, mean, s, min, max
  - type of distribution—normal, uniform, etc.
- Use a top-down approach to answer spatial data queries
- Start from a pre-selected layer—typically with a small number of cells
- For each cell in the current level compute the confidence interval

STING

- Remove the irrelevant cells from further consideration
- When finish examining the current layer, proceed to the next lower level
- Repeat this process until the bottom layer is reached
- Advantages:
  - Query-independent, easy to parallelize, incremental update
  - *O(K),* where *K* is the number of grid cells at the lowest level
- Disadvantages:
  - All the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected

Evaluation of Clustering

- Assess if non-random structure exists in the data by measuring the probability that the data is generated by a uniform data distribution
- Test spatial randomness by statistic test: Hopkins Statistic
  - Given a dataset D regarded as a sample of a random variable o, determine how far away o is from being uniformly distributed in the data space
  - Sample *n* points, $p_1, \ldots, p_n$, uniformly from D. For each p_i, find its nearest neighbor in D: $x_i = min\{dist(p_i, v)\}$ where *v* in D
  - Sample *n* points, $q_1, \ldots, q_n$, uniformly from D. For each q_i, find its nearest neighbor in D – $\{q_i\}$: $y_i = min\{dist(q_i, v)\}$ where *v* in D and $v \neq q_i$
  - Calculate the Hopkins Statistic:

$$H = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i}$$

  - If D is uniformly distributed, $\sum x_i$ and $\sum y_i$ will be close to each other and H is close to 0.5. If D is highly skewed, H is close to 0

Evaluation of Clustering

- Empirical method
  - # of clusters ≈√n/2 for a dataset of n points
- Elbow method
  - Use the turning point in the curve of sum of within cluster variance w.r.t the # of clusters
- Cross validation method
  - Divide a given data set into *m* parts
  - Use *m* – 1 parts to obtain a clustering model
  - Use the remaining part to test the quality of the clustering
    - E.g., For each point in the test set, find the closest centroid, and use the sum of squared distance between all points in the test set and the closest centroids to measure how well the model fits the test set
  - For any k > 0, repeat it *m* times, compare the overall quality measure w.r.t. different *k's*, and find # of clusters that fits the data the best

Evaluation of Clustering

- Two methods: extrinsic vs. intrinsic
- Extrinsic: supervised, i.e., the ground truth is available
  - Compare a clustering against the ground truth using certain clustering quality measure
  - Ex. Bcubed, precision and recall metrics
- Intrinsic: unsupervised, i.e., the ground truth is unavailable
  - Evaluate the goodness of a clustering by considering how well the clusters are separated, and how compact the clusters are
  - Ex. Silhouette coefficient

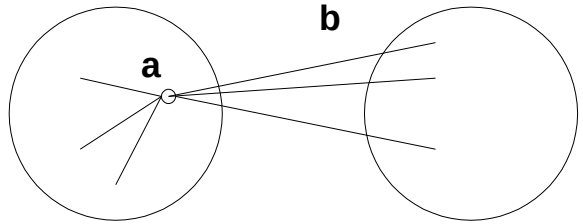
Evaluation of Clustering

- Clustering quality measure: $Q(C, C_g)$, for a clustering C given the ground truth C_g.
- Q is good if it satisfies the following **4** essential criteria
  - **Cluster homogeneity**: the purer, the better
  - **Cluster completeness**: should assign objects belong to the same category in the ground truth to the same cluster
  - **Rag bag**: putting a heterogeneous object into a pure cluster should be penalized more than putting it into a *rag bag* (i.e., "miscellaneous" or "other" category)
  - **Small cluster preservation**: splitting a small category into pieces is more harmful than splitting a large category into pieces

Silhouette Coefficient

- Silhouette Coefficient combines ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings
- For an individual point i
  - Calculate a_i = average distance of i to the points in its own cluster
  - Calculate b_i = min (over clusters) of the average distance of i to points in other clusters
  - The silhouette coefficient for a point i is then given by

$$s_i = 1 - a_i / b_i$$



  - Typically between 0 and 1, the closer to 1 the better.
  - Can be less than 0 but this is a problematic case

- Can calculate the Average Silhouette coefficient for a cluster, or for a clustering

Silhouette Coefficient

https://www.mathworks.com/help/stats/silhouette.html

Silhouette Coefficient

Cluster 1 ={{1,0},{1,1}}
Cluster 2 ={{1,2},{2,3},{2,2},{1,2}},
Cluster 3 ={{3,1},{3,3},{2,1}}

Take a point {1,0} in cluster 1
Calculate its **average distance** to all other points in **it's cluster**, i.e. cluster 1

So a1 =√((1-1)^2 + (0-1)^2) =√(0+1)=√1=1

Now for the object {1,0} in cluster 1 calculate its average distance from all the objects in cluster 2 and cluster 3.
Of these take the **minimum** average distance.

So for cluster 2
{1,0} ----> {1,2} = distance = √((1-1)^2 + (0-2)^2) =√(0+4)=√4=2
{1,0} ----> {2,3} = distance = √((1-2)^2 + (0-3)^2) =√(1+9)=√10=3.16
{1,0} ----> {2,2} = distance = √((1-2)^2 + (0-2)^2) =√(1+4)=√5=2.24
{1,0} ----> {1,2} = distance = √((1-1)^2 + (0-2)^2) =√(0+4)=√4=2

Silhouette Coefficient

Cluster 1 ={{1,0},{1,1}}
Cluster 2 ={{1,2},{2,3},{2,2},{1,2}},
Cluster 3 ={{3,1},{3,3},{2,1}}

Therefore, the average distance of point {1,0} in cluster 1 to all the points in cluster 2 =
(2+3.16+2.24+2)/4 = 2.325

Similarly, for cluster 3
{1,0} ----> {3,1} = distance = √((1-3)^2 + (0-1)^2) =√(4+1)=√5=2.24
{1,0} ----> {3,3} = distance = √((1-3)^2 + (0-3)^2) =√(4+9)=√13=3.61
{1,0} ----> {2,1} = distance = √((1-2)^2 + (0-1)^2) =√(1+1)=√2=2.24

Therefore, the **average distance** of point {1,0} in cluster 1 to all the points in cluster 3 =
(2.24+3.61+2.24)/3 = 2.7

Now, the **minimum** average distance of the point {1,0} in cluster 1 to the other clusters 2 and 3 is,
b1 =2.325 (2.325 < 2.7)

Silhouette Coefficient

Cluster 1 ={{1,0},{1,1}}
Cluster 2 ={{1,2},{2,3},{2,2},{1,2}},
Cluster 3 ={{3,1},{3,3},{2,1}}

So the silhouette coefficient of cluster 1
s1= 1-(a1/b1) = 1- (1/2.325)=1-0.4301=0.5699

In a similar fashion you need to calculate the silhouette coefficient for cluster 2 and cluster 3 separately by taking any single object point in each of the clusters and repeating the steps above.

Of these the cluster with the greatest silhouette coefficient is the best as per evaluation.

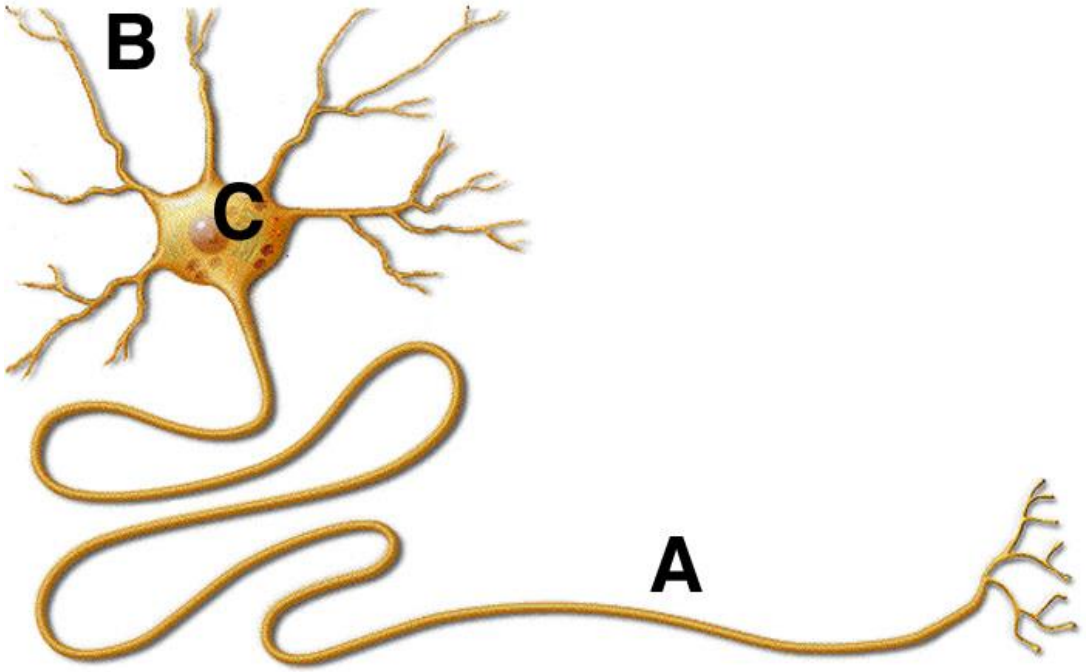
Artificial Neural Networks

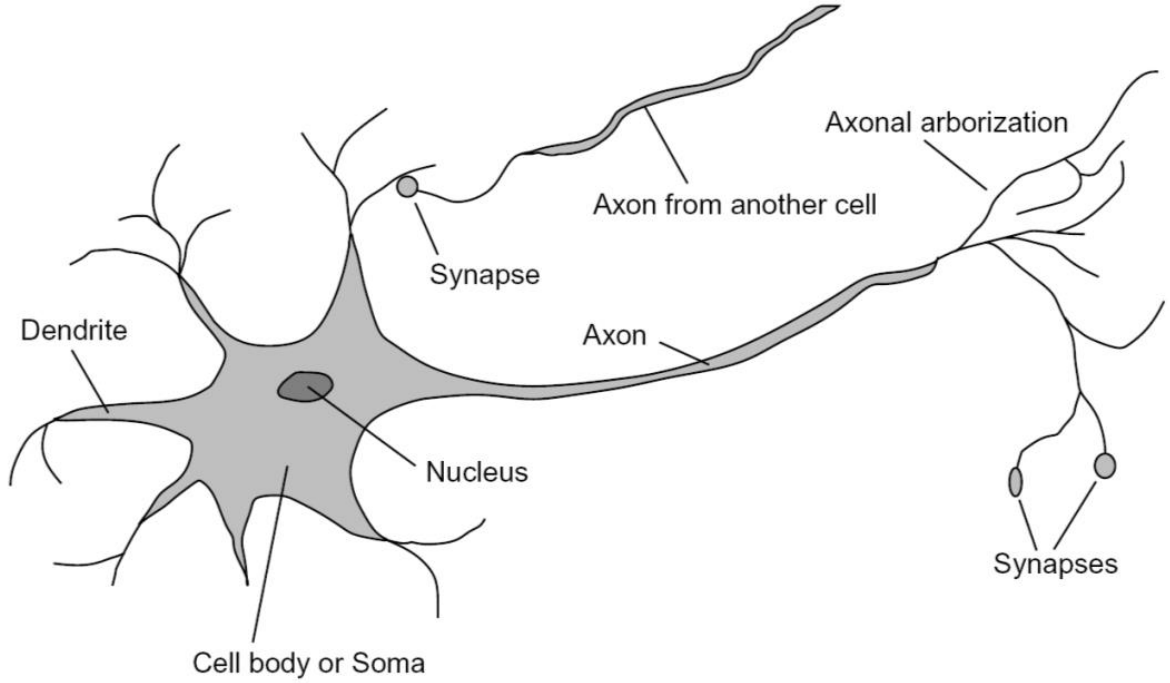
Adopted from Garrison Cotrell - Darwish Ashraf - Nielsen Morten – ‘Data Mining: Concepts & Techniques’

Introduction

- We have billions and billions of neurons that somehow work together to create the mind.
- These neurons are connected by 10^{14} - 10^{15} synapses, which we think encode the "knowledge" in the network - too many for us to explicitly program them in our models
- Rather we need some way to ***indirectly*** set them via a procedure that will achieve some goal by changing the synaptic strengths (which we call weights).
- This is called ***learning*** in these systems.

Introduction





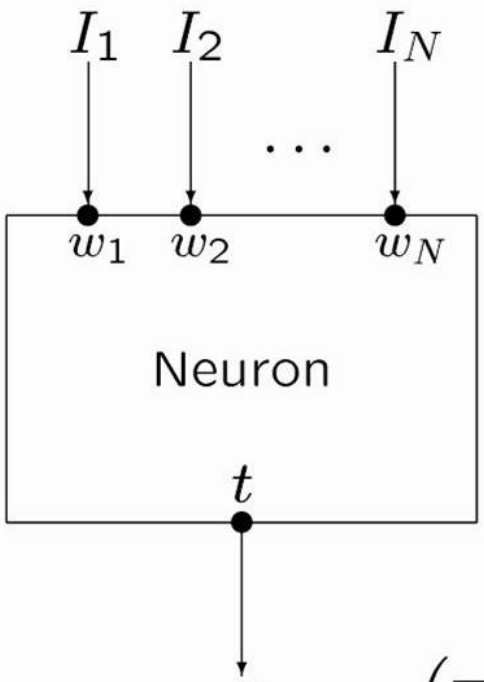
Introduction

Input signals

Synaptic weights

Threshold

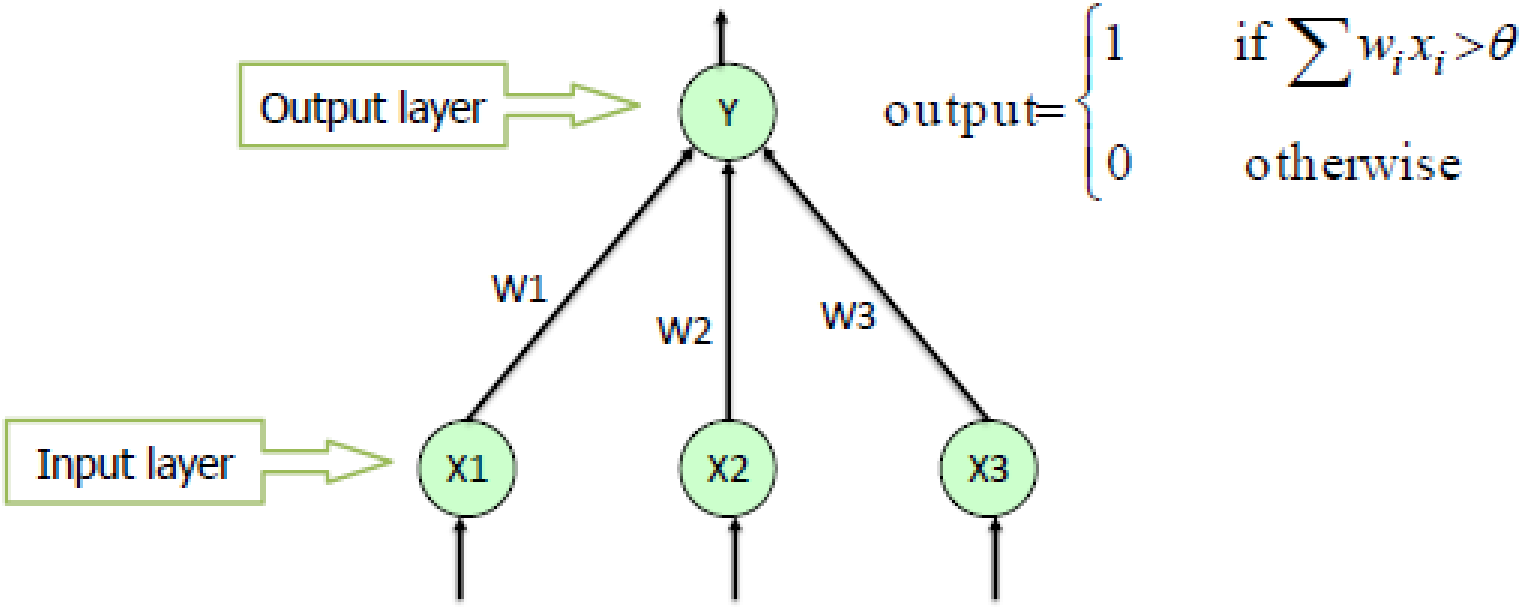
Output signal



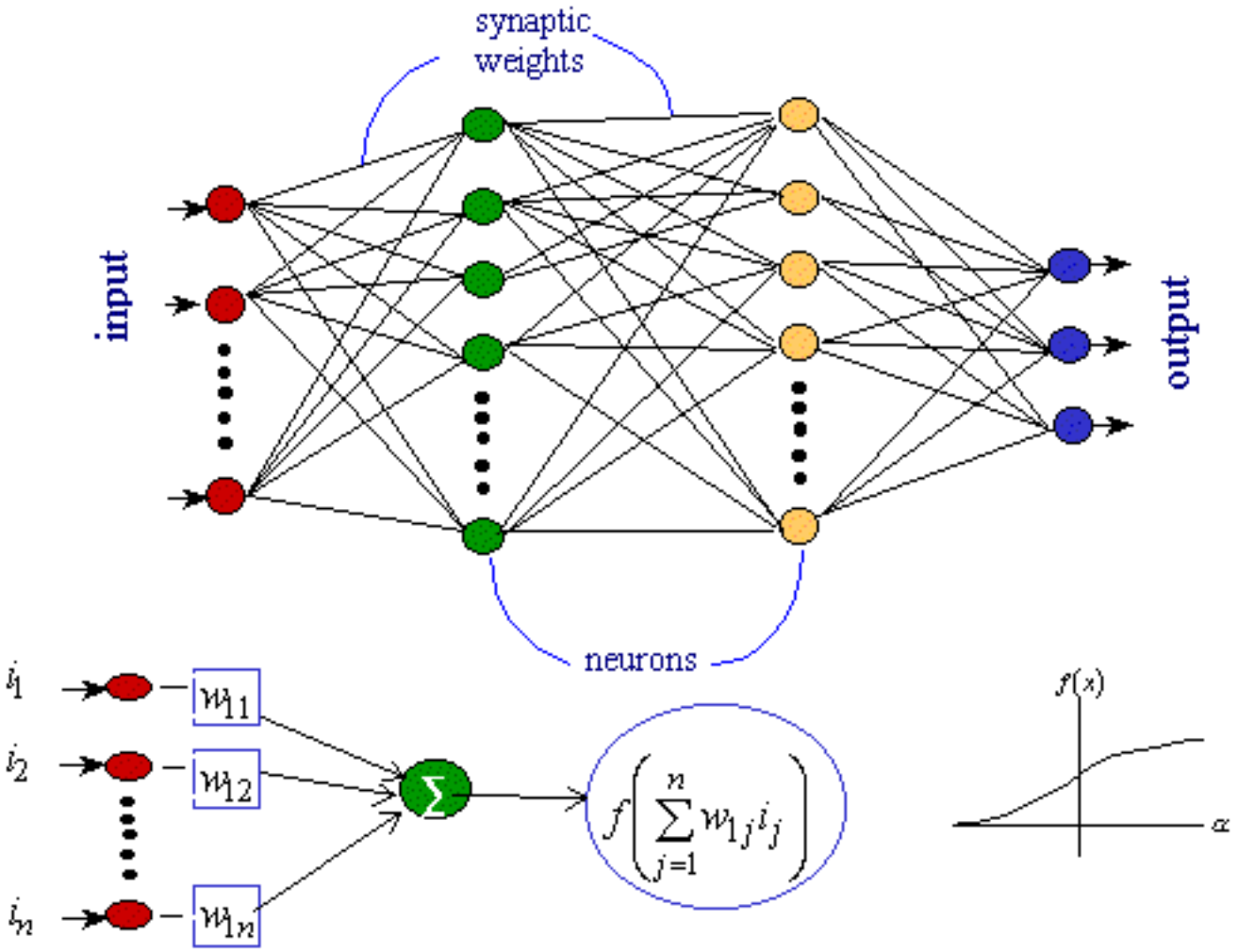
$$O = \sigma\left(\sum_{n=1}^{N} w_n I_n - t\right)$$

Perceptron

Single Layer Perceptron



Multilayer Perceptrons

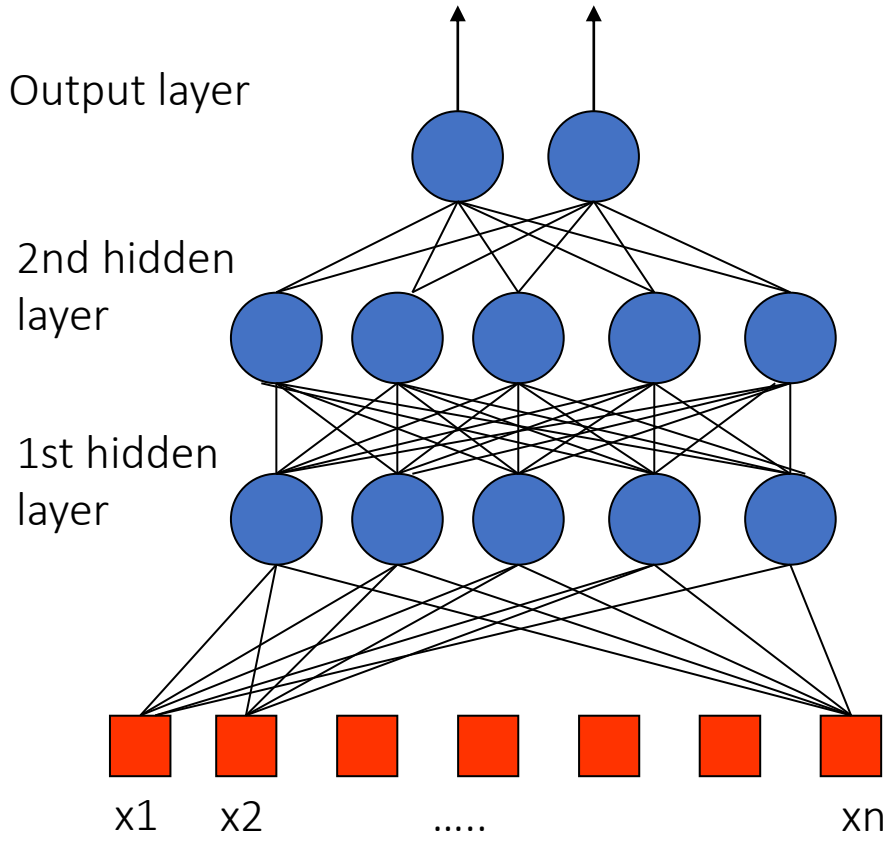


Calculations

- A mathematical model to solve engineering problems
  - Group of highly connected neurons to realize compositions of non linear functions
- Tasks
  - Classification
  - Discrimination
  - Estimation
- 2 types of networks
  - **Feed forward Neural Networks**
  - **Recurrent Neural Networks**

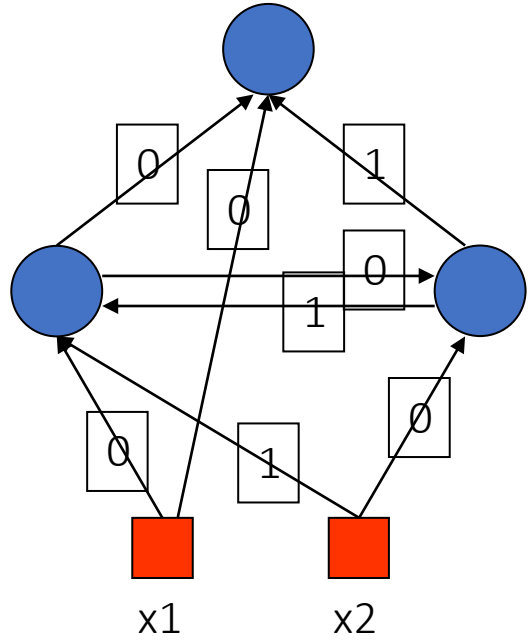
Feed Forward Neural Networks

- The information is propagated from the inputs to the outputs
- Computations of No non linear functions from n input variables by compositions of Nc algebraic functions
- Time has no role (NO cycle between outputs and inputs)



Recurrent Neural Networks

- Can have arbitrary topologies
- Can model systems with internal states (dynamic ones)
- Delays are associated to a specific weight
- Training is more difficult
- Performance may be problematic
  - Stable Outputs may be more difficult to evaluate
  - Unexpected behavior (oscillation, chaos, …)



Learning

- The procedure that consists in estimating the parameters of neurons so that the whole network can perform a specific task

- 2 types of learning
  - **The supervised learning**
  - **The unsupervised learning**

- The Learning process (supervised)
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs

Supervised Learning

- The desired response of the neural network in function of particular inputs is well known.
- A “Professor” may provide examples and teach the neural network how to fulfill a certain task

Unsupervised Learning

- Idea : group typical input data in function of resemblance criteria un-known a priori
- Data clustering
- No need of a professor
  - The network finds itself the correlations between the data
  - Examples of such networks :
    - Kohonen feature maps

Learning

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Pros & Cons

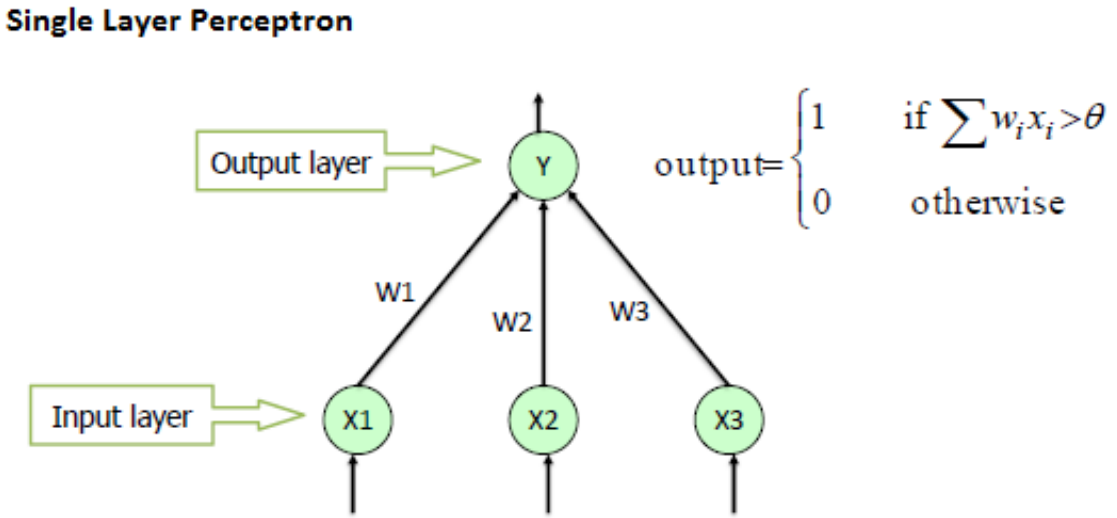
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - **Poor interpretability**: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Architectures

- Perceptron
- Multi-Layer Perceptron
- Radial Basis Function (RBF)
- Kohonen Features maps
- Other architectures
  - An example : Shared weights neural networks

Perceptron

- Rosenblatt (1962) discovered a learning rule for perceptrons called the ***perceptron convergence procedure***.
- Guaranteed to learn anything computable (by a two-layer perceptron)
- Unfortunately, not everything was computable (Minsky & Papert, 1969)



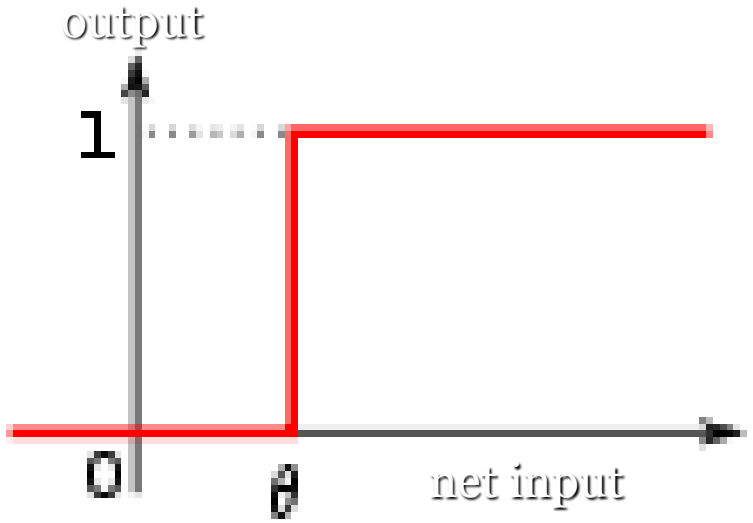
Perceptron

- Output activation rule:
  - First, compute the *net input* to the output unit:

    $\Sigma_i w_i x_i = net$
  - Then, compute the output as:

    If $net \geq \theta$ then output = 1

    else output = 0



Perceptron

- Perceptrons only be 100% accurate only on linearly separable problems.
- **Multi-layer networks (often called *multi-layer perceptrons*, or *MLPs*) can represent any target function**.
- However, in multi-layer networks, there is no guarantee of convergence to minimal error weight vector.

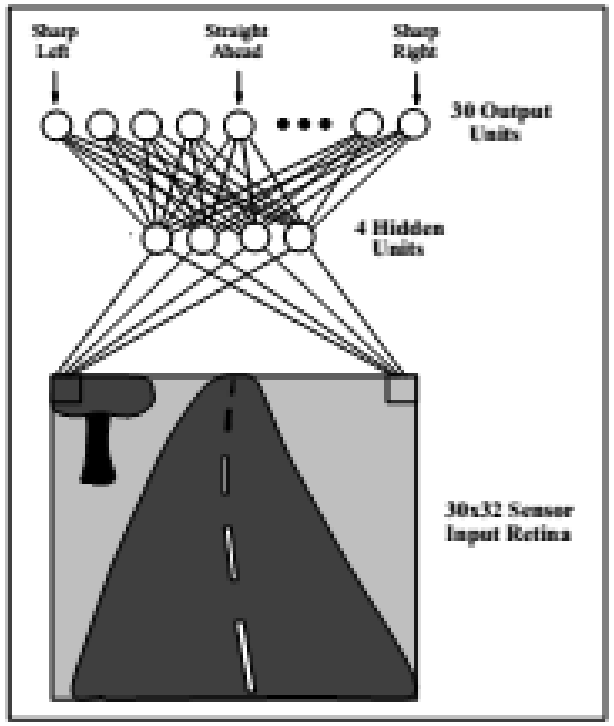
Perceptron

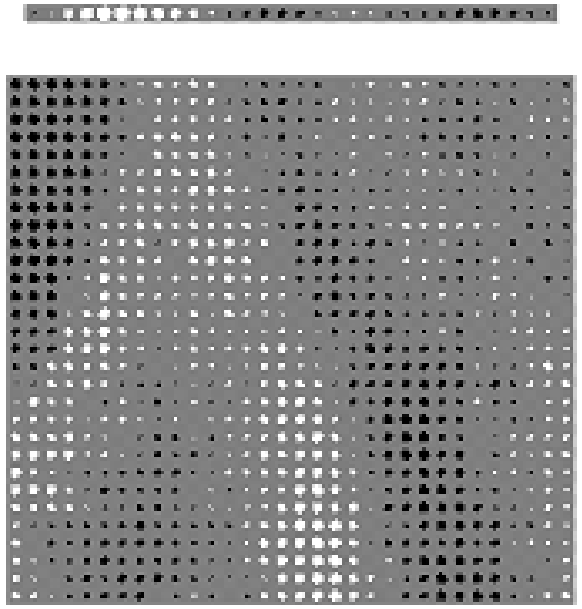
- Learning rule:
  - If output is 1 and should be 0, then *lower* weights to active inputs and *raise* the threshold θ
  - If output is 0 and should be 1, then *raise* weights to active inputs and *lower* the threshold θ

("active input" means $x_i = 1$, not 0)

Perceptron

- Each output unit correspond to a particular steering direction.
- The most highly activated one gives the direction to steer.





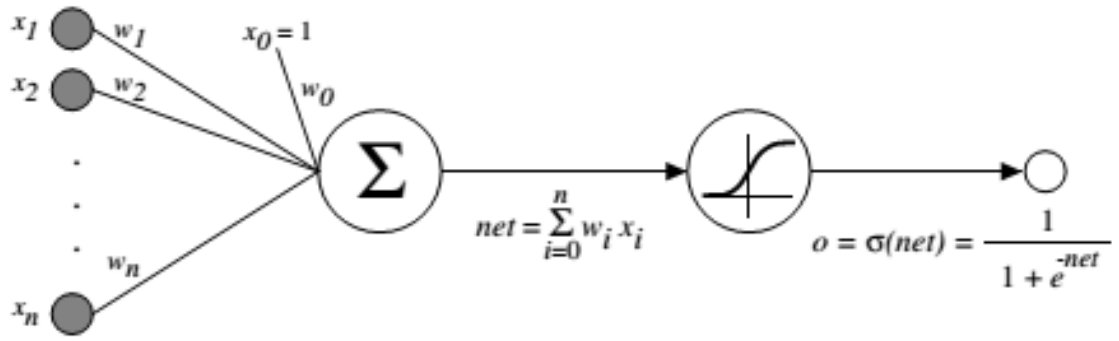
(Note: bias units and weights not shown)

Perceptron

- Before 2009: ANNs typically with 2-3 layers
  - Reason 1: computation times
  - Reason 2: problems of the backpropagation algorithm
    - Local optimization only (needs a good initialization, or re-initialization)
    - Prone to over-fitting (too many parameters to estimate, too few labeled examples)
  - => Skepticism: A deep network often performed worse than a shallow one
- After 2009: Deep neural networks
  - Fast GPU-based implementations
  - Weights can be initialized better (Use of unlabeled data, Restricted Boltzmann Machines)
  - Large collections of labeled data available
  - Reducing the number of parameters by weight sharing
  - Improved backpropagation algorithm
  - Success in different areas, e.g. traffic sign recognition, handwritten digits problem

Perceptron

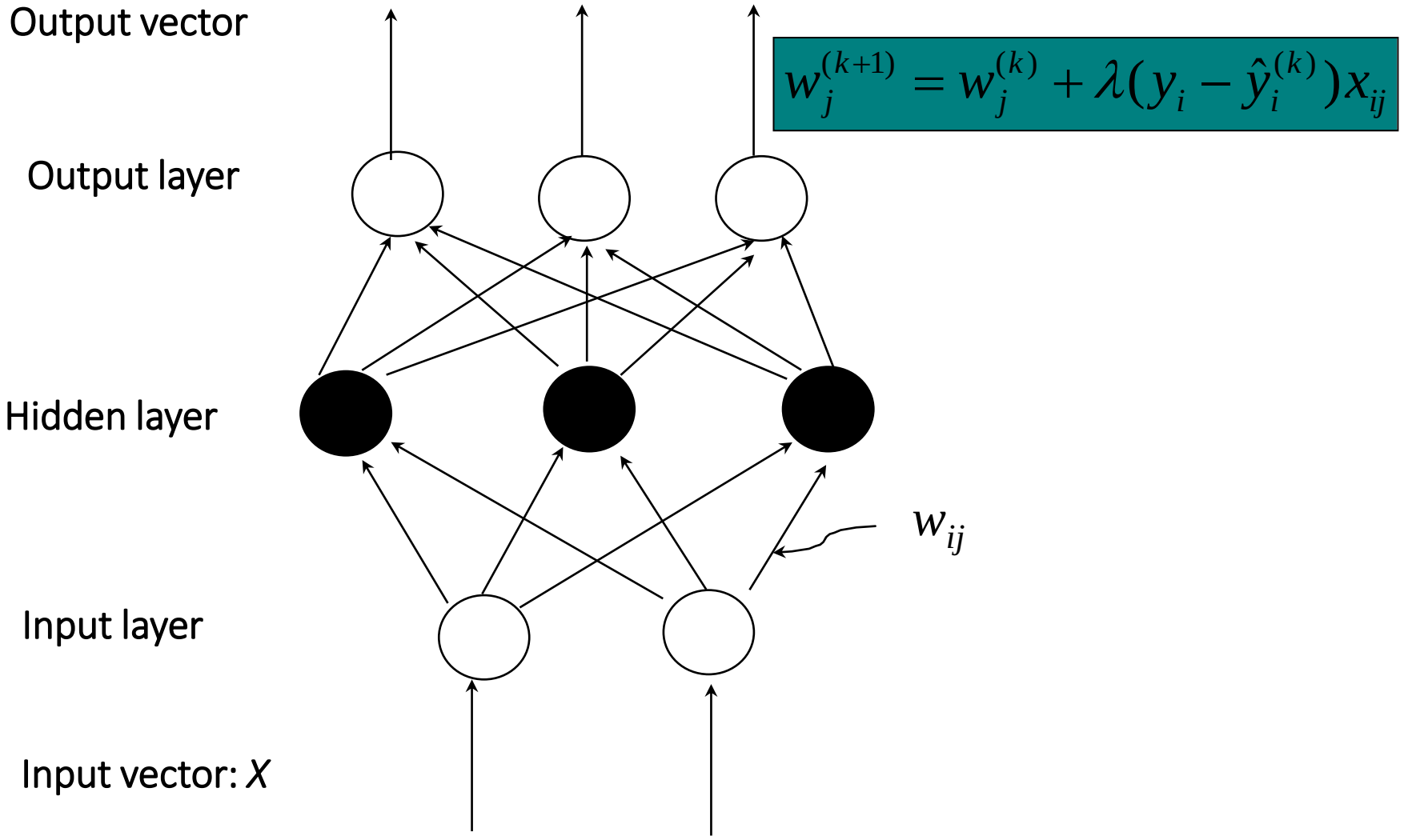
- Network node in detail



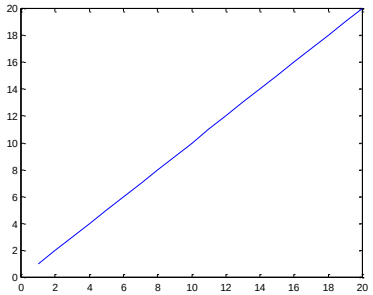
$\sigma(x)$ is the sigmoid function

- Network learning process = tuning the synaptic weights
  - Initialize randomly
  - Repeatedly compute the ANN result for a given task, compare with ground truth, update ANN weights by *backpropagation* algorithm to improve ANN performance

Perceptron

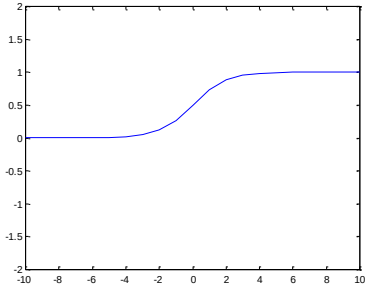

Output vector
Output layer
Hidden layer
Input layer
Input vector: X
$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
w_{ij}

Perceptron



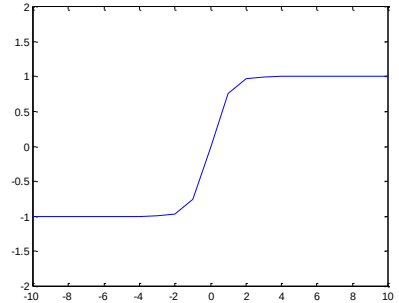
Linear

$$y = x$$



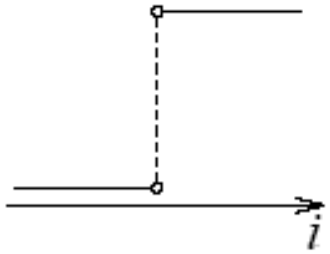
Logistic

$$y = \frac{1}{1+\exp(-x)}$$

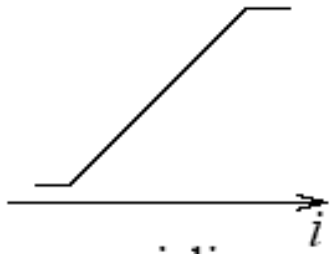


Hyperbolic tangent

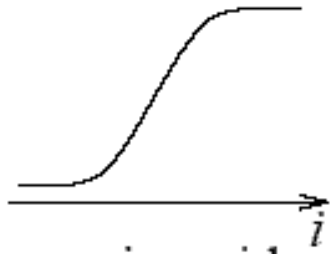
$$y = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



sgn



semi-linear

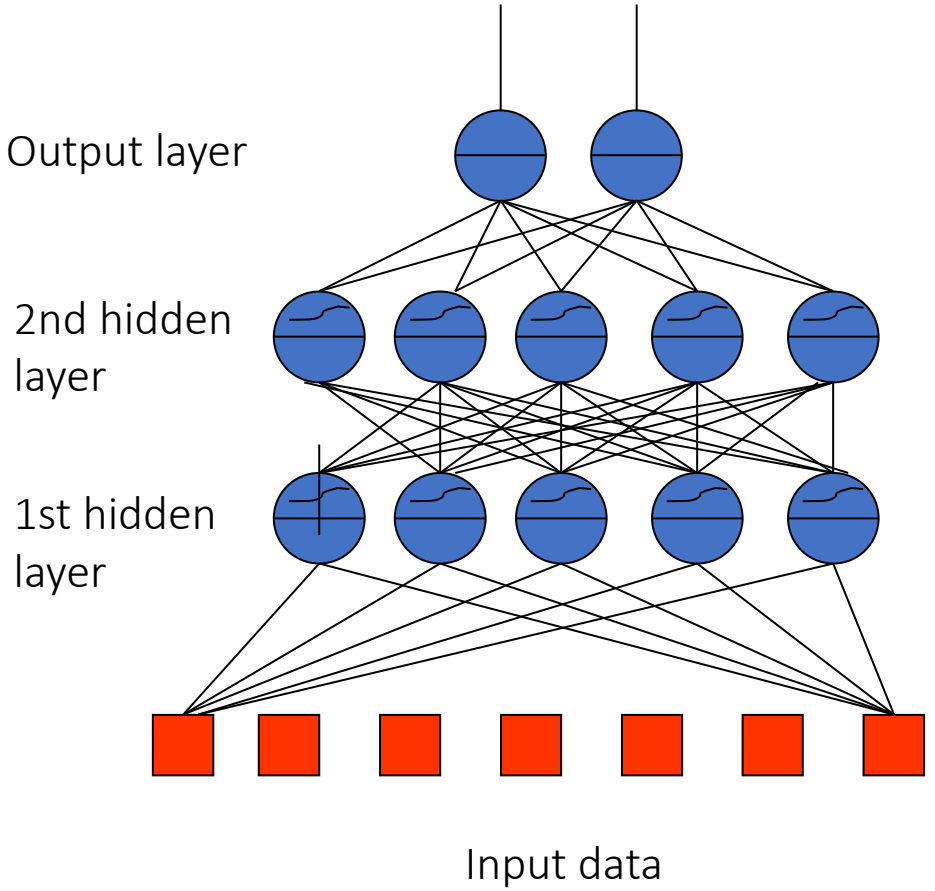


sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Multi-Layer Perceptrons

- One or more hidden layers
- Sigmoid activations functions



Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Backpropagation

$$net_j = w_{j0} + \sum_i^n w_{ji} o_i$$

$$o_j = f_j(net_j)$$

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}} = -\alpha \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \alpha \delta_j o_i$$

Credit assignment

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\delta_j = -\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -\frac{\partial E}{\partial o_j} f'(net_j)$$

$$E = \frac{1}{2}(t_j - o_j)^2 \Rightarrow \frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

$$\delta_j = (t_j - o_j) f'(net_j)$$

If the jth node is an output unit

Backpropagation (differentiable perceptron)

Define total classification error or loss on the training set:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j)\right)^2, \; f_{\mathbf{w}}(\mathbf{x}_j) = \sigma(\mathbf{w} \cdot \mathbf{x}_j), \;\; \sigma(t) = \frac{1}{1+e^{-t}}$$

Update weights by *gradient descent*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right) \sigma'(\mathbf{w} \cdot \mathbf{x}_j) \frac{\partial}{\partial \mathbf{w}} (\mathbf{w} \cdot \mathbf{x}_j) \right]$$

$$= \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right) \sigma(\mathbf{w} \cdot \mathbf{x}_j)(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_j))\mathbf{x}_j \right]$$

For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - f(\mathbf{x})\right) \sigma(\mathbf{w} \cdot \mathbf{x})(1 - \sigma(\mathbf{w} \cdot \mathbf{x}))\mathbf{x}$$

Backpropagation (differentiable perceptron)

- For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\sigma(\mathbf{w}\cdot\mathbf{x})(1-\sigma(\mathbf{w}\cdot\mathbf{x}))\mathbf{x}$$

- Compare with update rule with non-differentiable perceptron:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\mathbf{x}$$

Example

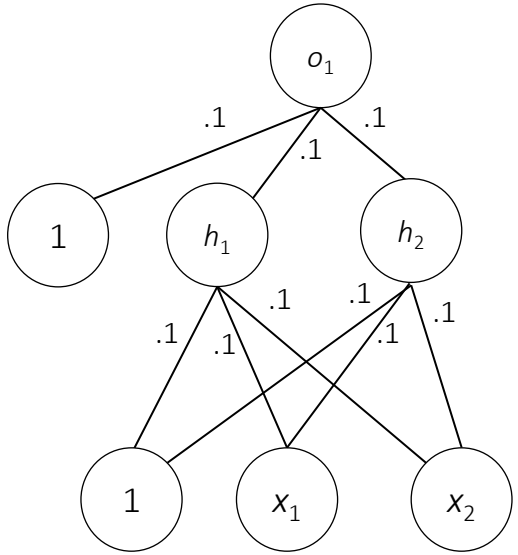
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Example

Training set:

1 0 Label: Positive

0 1 Label: Negative



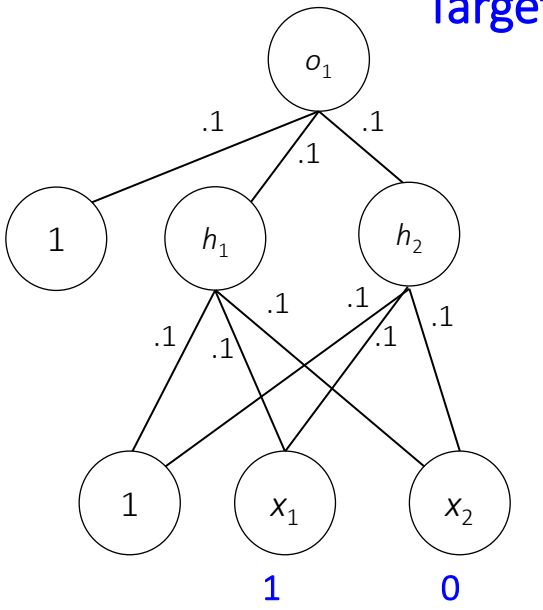
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Label: Positive

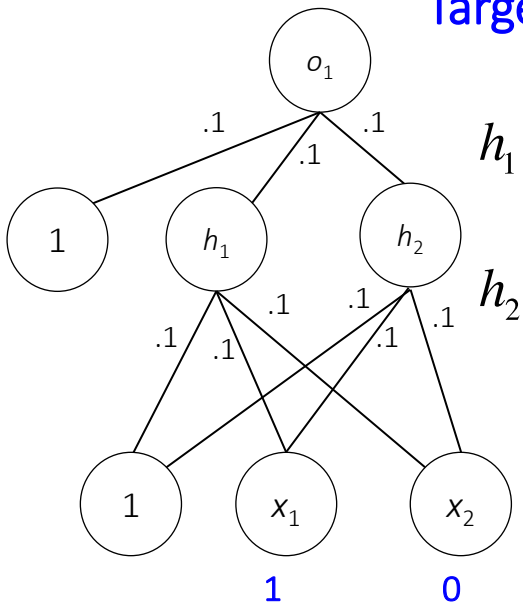
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



$$h_1 = \sigma\left((1)(.1)+(1)(.1)+(0)(.1)\right) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

$$h_2 = \sigma\left((1)(.1)+(1)(.1)+(0)(.1)\right) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

Label: Positive

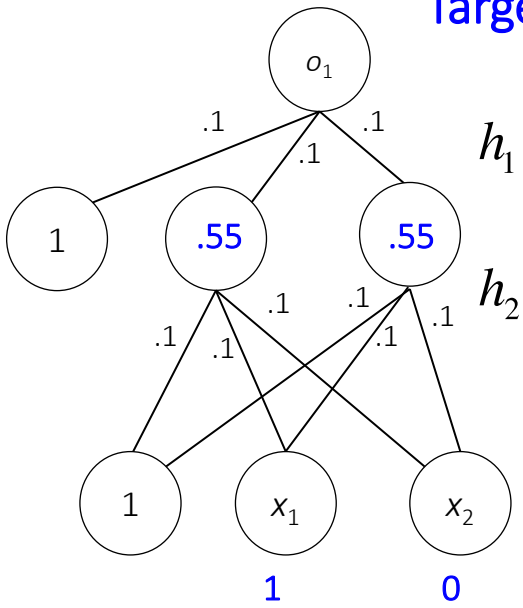
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



$$h_1 = \sigma\left((1)(.1)+(1)(.1)+(0)(.1)\right) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

$$h_2 = \sigma\left((1)(.1)+(1)(.1)+(0)(.1)\right) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

Label: Positive

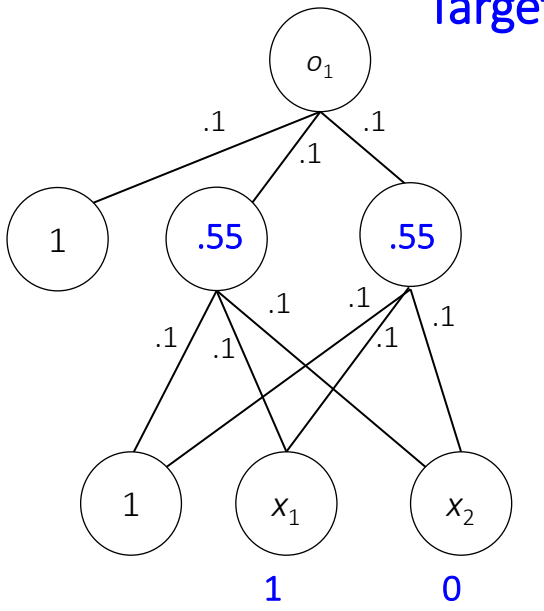
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



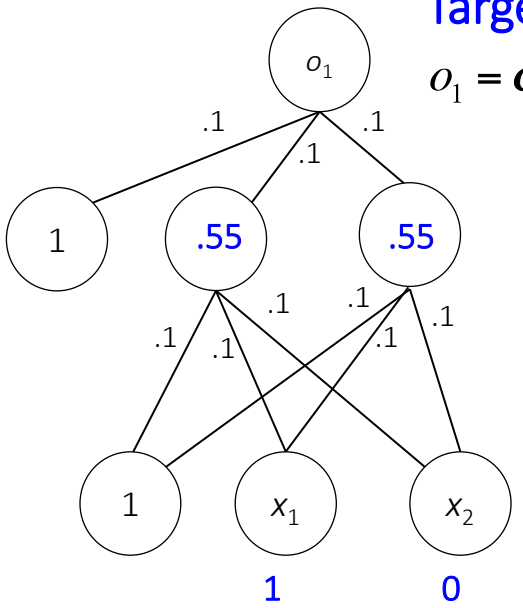
Label: Positive

Example

Training set:

1 0 Label: Positive

0 1 Label: Negative



Target: .9

$$o_1 = \sigma\big((1)(.1)+(.55)(.1)+(.55)(.1)\big) = \sigma(.21) = \frac{1}{1+e^{-.21}} = .552$$

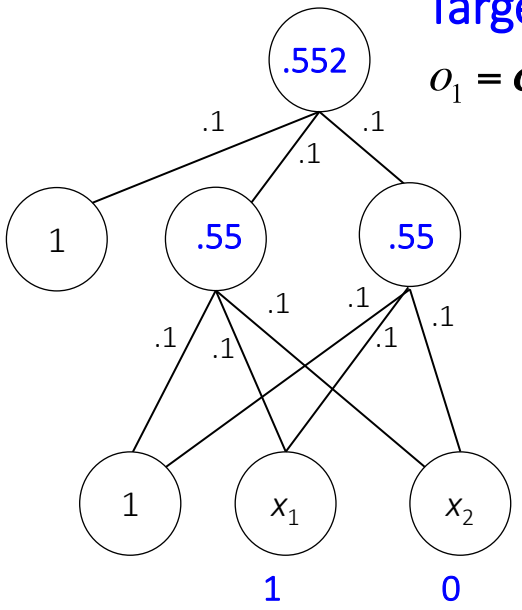
Label: Positive

Example

Training set:

1 0 Label: Positive

0 1 Label: Negative



Target: .9

$$o_1 = \sigma\big((1)(.1)+(.55)(.1)+(.55)(.1)\big) = \sigma(.21) = \frac{1}{1+e^{-.21}} = .552$$

Label: Positive

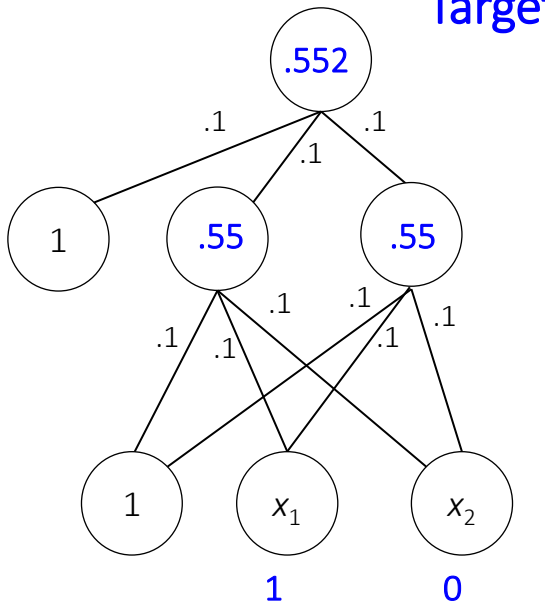
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Here we interpret $o_1 > .5$ as "positive".

Classification is correct.

But we still update weights.

Label: Positive

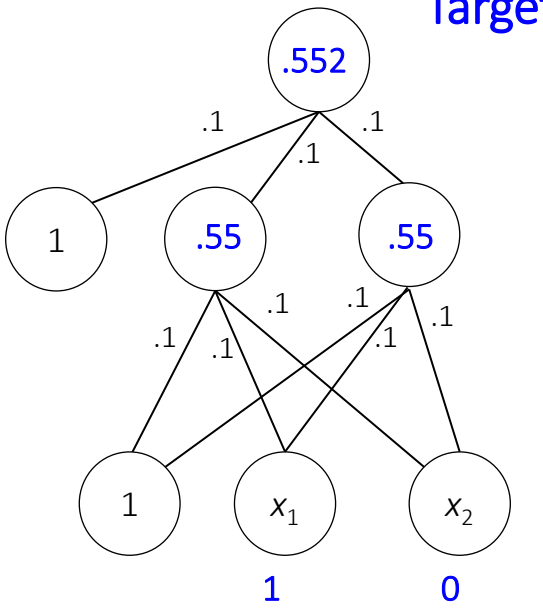
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

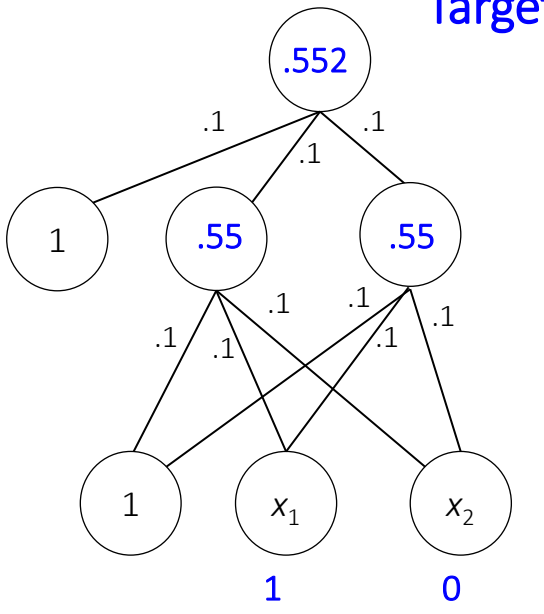
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

Update hidden-to-output weights (*learning rate* = 0.2; *momentum* = 0.9):

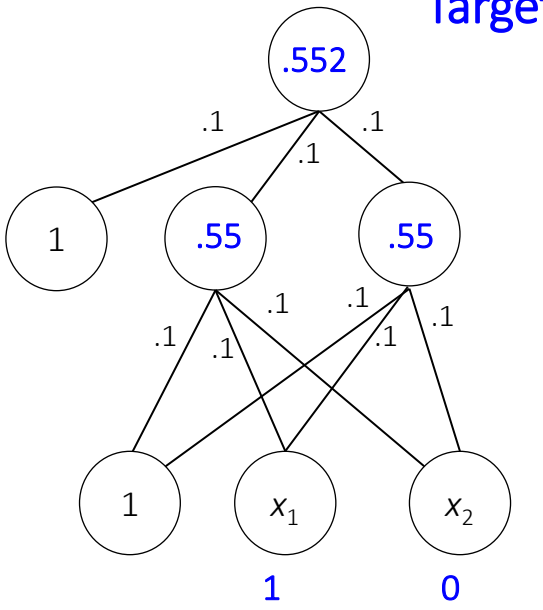
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

Update hidden-to-output weights (*learning rate* = 0.2; *momentum* = 0.9):

$$\Delta w^1_{k=1,j=0} = (.2)(.086)(1) + (.9)(0) = .0172$$

$$\Delta w^1_{k=1,j=1} = (.2)(.086)(.55) + (.9)(0) = .0095$$

$$\Delta w^1_{k=1,j=2} = (.2)(.086)(.55) + (.9)(0) = .0095$$

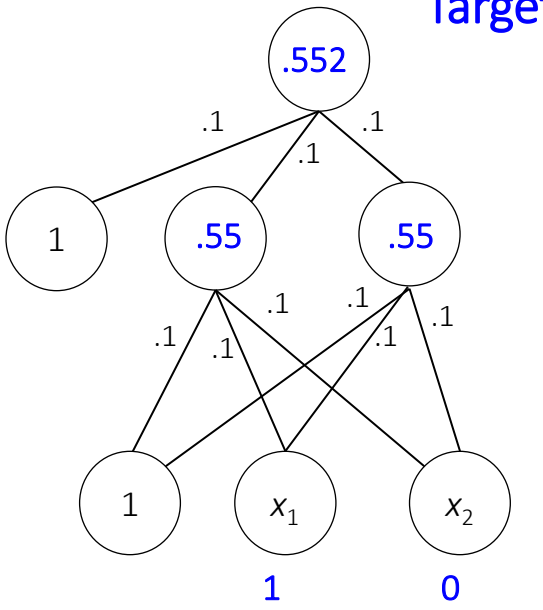
Training set:

1 0 Label: Positive

0 1 Label: Negative

Example

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

Update hidden-to-output weights (learning rate = 0.2; momentum = 0.9):

$$\Delta w^1_{k=1,j=0} = (.2)(.086)(1) + (.9)(0) = .0172 \qquad w^1_{k=1,j=0} = .1 + .0172 = .1172$$

$$\Delta w^1_{k=1,j=1} = (.2)(.086)(.55) + (.9)(0) = .0095 \qquad w^1_{k=1,j=1} = .1 + .0095 = .1095$$

$$\Delta w^1_{k=1,j=2} = (.2)(.086)(.55) + (.9)(0) = .0095 \qquad w^1_{k=1,j=2} = .1 + .0095 = .1095$$

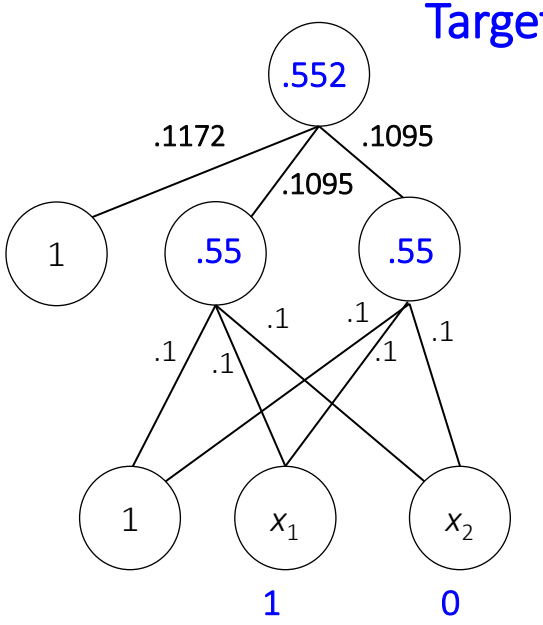
Training set:

1 0 Label: Positive

0 1 Label: Negative

Example

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9-.552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

$$\Delta w^1_{k=1,j=0} = (.2)(.086)(1)+(.9)(0) = .0172 \qquad w^1_{k=1,j=0} = .1+.0172 = .1172$$

$$\Delta w^1_{k=1,j=1} = (.2)(.086)(.55)+(.9)(0) = .0095 \qquad w^1_{k=1,j=1} = .1+.0095 = .1095$$

$$\Delta w^1_{k=1,j=2} = (.2)(.086)(.55)+(.9)(0) = .0095 \qquad w^1_{k=1,j=2} = .1+.0095 = .1095$$

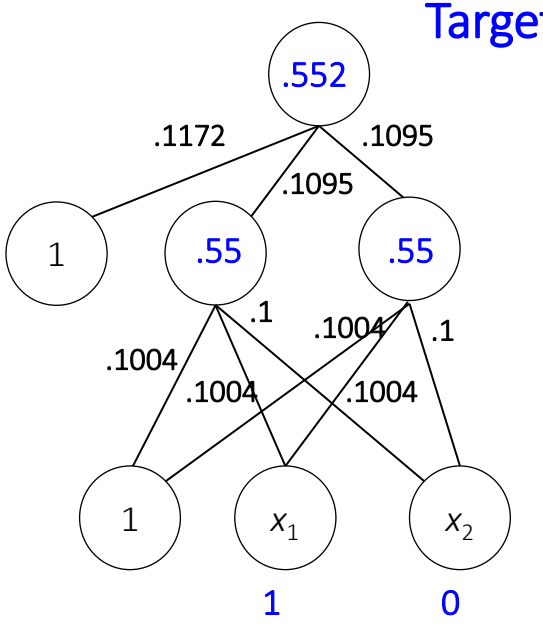
Training set:

1 0 Label: Positive

0 1 Label: Negative

Example

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

Label: Positive

Update input-to-hidden weights (*learning rate* = 0.2; *momentum* = 0.9):

$$\Delta w^1_{j=1,i=0} = (.2)(.002)(1) + (.9)(0) = .0004 \qquad w^1_{j=1,i=0} = .1 + .0004 = .1004$$

$$\Delta w^1_{j=1,i=1} = (.2)(.002)(1) + (.9)(0) = .0004 \qquad w^1_{j=1,i=1} = .1 + .0004 = .1004$$

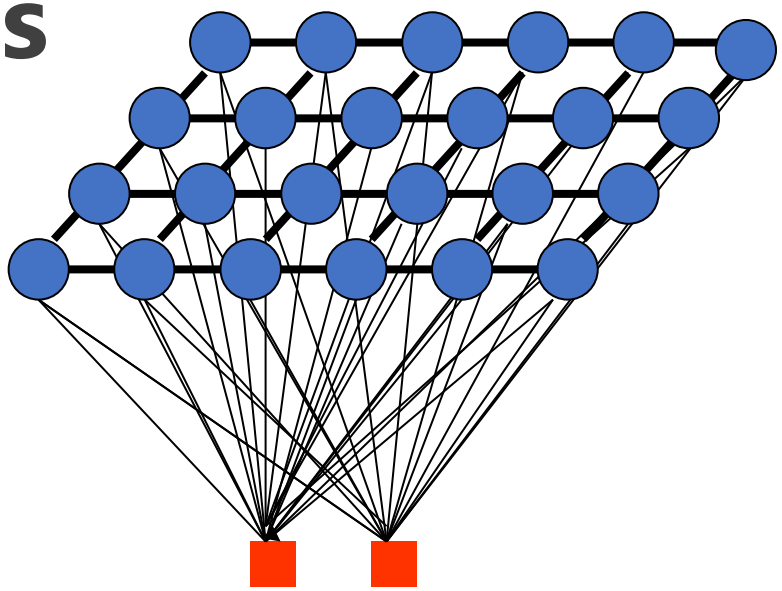
$$\mathrm{D}w^1_{j=1,i=2} = (.2)(.002)(0) + (.9)(0) = 0 \qquad w^1_{j=1,\,i=2} = .1 \qquad w^1_{j=2,\,i=2} = .1$$

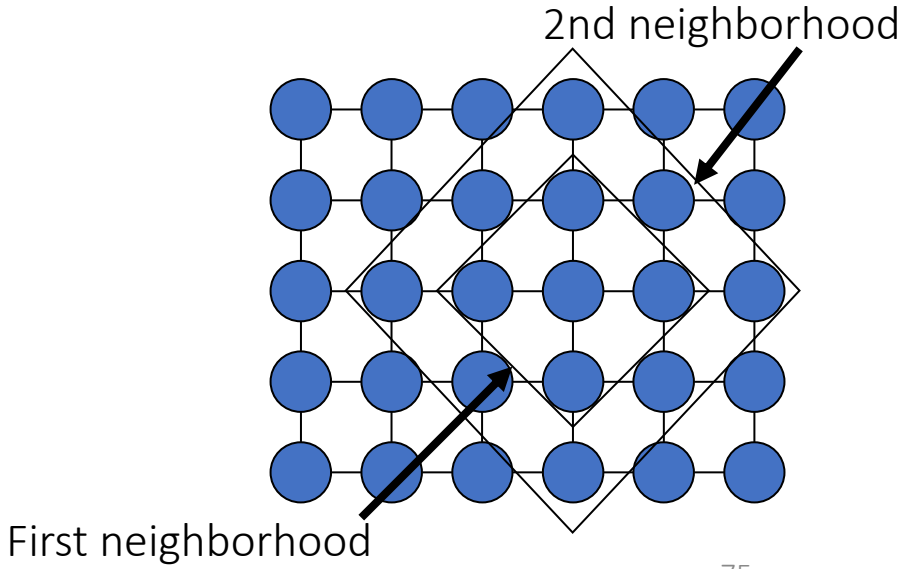
Self Organizing Maps

- The purpose of SOM is to map a multidimensional input space onto a topology preserving map of neurons
  - Preserve a topological so that neighboring neurons respond to « similar »input patterns
  - The topological structure is often a 2 or 3 dimensional space
- Each neuron is assigned a weight vector with the same dimensionality of the input space
- Input patterns are compared to each weight vector and the closest wins (Euclidean Distance)

Self Organizing Maps

- The activation of the neuron is spread in its direct neighborhood =>neighbors become sensitive to the same input patterns
- Block distance
- The size of the neighborhood is initially large but reduce over time => Specialization of the network





Self Organizing Maps

- During training, the “winner” neuron and its neighborhood adapts to make their weight vector more similar to the input pattern that caused the activation
- The neurons are moved closer to the input pattern
- The magnitude of the adaptation is controlled via a learning parameter which decays over time