Neural Networks 1 - Self-organization

18NES1 - Lecture 12, Summer semester 2024/25

Zuzana Petříčková

May 6th, 2025

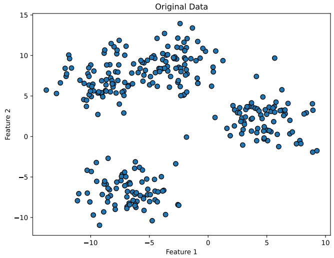
This Week

Unsupervised Learning (Self-Organization)

- Clustering and the k-Means Algorithm
- Other clustering algorithms: hierarchical clustering
- Single-layer neural network and competitive learning (e.g., online k-means)
- Self-organising feature maps (SOM)
- Demonstrations and examples

Unsupervised Learning and Self-Organization

- Training set T in the form $T = \{\vec{x}_1, \ldots, \vec{x}_N\}$ (only inputs)
- $\vec{x}_i \in \mathbb{R}^n$ is the i-th training input pattern, target outputs are unknown
- **Idea:** the model itself decides which response is best for a given input and adjusts its weights accordingly $\rightarrow$ self-organization

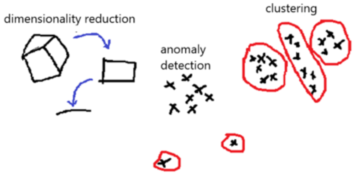


- We have data but no knowledge of its internal structure
- The goal is to uncover the structure and patterns within the data

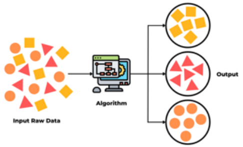
Unsupervised Learning and Self-Organization

- **Goal:** to discover structure or patterns within the data
- **Applications:**
    - **Dimensionality reduction** (data compression, visualization)
    - **Anomaly detection** (e.g., in banking transactions)
    - **Clustering** (e.g., customer segmentation, plagiarism detection)
    - E-commerce: recommendation systems

Types of Tasks:



https://towardsdatascience.com/unsupervised-learning-algorithms-cheat-sheet-d391a39de44a



https://eastgate-software.com/what-is-unsupervised-learning/#ftoc-heading-7

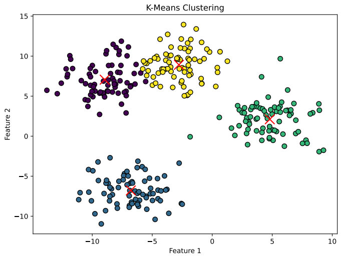
Unsupervised Learning and Self-Organization

Cluster

- A group of samples with **high similarity among themselves** and **low similarity to samples in other clusters**
- In simplified terms: **similarity = proximity**

Clustering

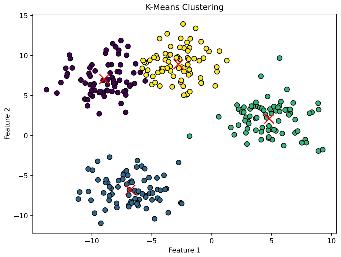
- Disjoint partitioning of data into clusters



Clustering

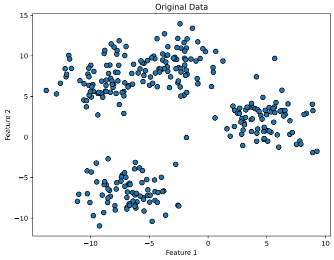
Challenges:

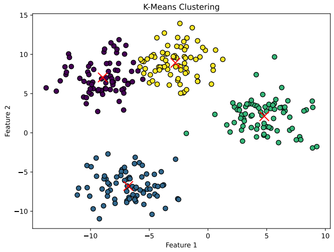
- How to determine the number and distribution of clusters in the feature space?
- How to choose the representative(s) of a cluster?
    - Appropriately selected training samples belonging to a cluster
    - Example: the centroid of a cluster



The k-Means Clustering Algorithm

- Unsupervised learning
- Input patterns are classified into k different clusters, each cluster l is represented by its centroid $\vec{c}_l$
- A new vector $\vec{x}$ is assigned to the cluster i whose centroid $\vec{c}_l$ is closest to it





The k-Means Clustering Algorithm

1. Given a training set $T = \{\vec{x}_1, ..., \vec{x}_N\}, \vec{x}_i \in \mathbb{R}^n$
2. Select k random vectors $\vec{c}_l, l = 1, ..., k$ (from $\mathbb{R}^n$ or from T) as initial cluster centroids
3. Repeat:
   - Assign each vector from T to the nearest cluster centroid
   - Recalculate the cluster centroids based on assigned patterns:

     $$\vec{c}_l = \frac{1}{n_l} \sum_{l_i=1}^{n_l} (\vec{x}_{l_i})$$

     where n_l is the number of vectors assigned to cluster l, and l_i indexes vectors assigned to cluster l
   - Repeat the above steps until the cluster memberships of training patterns no longer change

Parameters of the k-Means Algorithm

- Number of clusters k - How to set it?
- Distance metric - How to compute the distance (similarity) between numerical vectors?
- Initialization method - How to initialize positions of the centroids?
- Stopping criteria - When to stop trraining?

Parameters of the k-Means Algorithm

How to compute the distance (similarity) between numerical vectors?

- **Euclidean distance:** $d(\vec{p},\vec{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
- When only comparing distances (for efficiency), it is common to use the squared distance: $d(\vec{p},\vec{q}) = \sum_{i=1}^{n}(p_i - q_i)^2$

Other distance metrics include:

- **Manhattan (city block) distance:** $d(\vec{p},\vec{q}) = \sum_{i=1}^{n}|p_i - q_i|$
- **Chebyshev distance:** $d(\vec{p},\vec{q}) = \max_i |p_i - q_i|$ *"What is the biggest problem?"*
- **Minkowski distance:** $d(\vec{p},\vec{q}) = \left(\sum_{i=1}^{n}|p_i - q_i|^r\right)^{\frac{1}{r}}$ *Generalizes the previous metrics ($r = 2$, $r = 1$, $r \to \infty$)*
- **Cosine similarity:** $\cos(\vec{p},\vec{q}) = \frac{\vec{p}\cdot\vec{q}}{\|\vec{p}\|\|\vec{q}\|}$ *We focus on the direction, not the magnitude (useful for text processing)*
- (and others)

Parameters of the k-Means Algorithm

Initialization in k-Means

- The result of k-means depends heavily on the initial choice of centroids.
- Poor initialization $\rightarrow$ poor clustering, slow convergence, getting stuck in a local minimum.
- Initialization options:
  - Random vectors in $\mathbb{R}^n$ or within the range of the data
  - Random selection of points from the training set T
  - **k-means++**:
    - First centroid is chosen randomly
    - Subsequent centroids are chosen with probability proportional to the square of the distance to the nearest already chosen centroid
  - Running the algorithm multiple times and selecting the best solution (lowest sum of squared distances)

Parameters of the k-Means Algorithm

- Number of clusters k (usually specified by the user)
- Distance metric (default: Euclidean)
- Initialization method (random, k-means++, custom choice)
- Stopping criteria:
    - until cluster memberships stop changing
    - until centroids stop changing
    - reaching a maximum number of iterations
- Further improvements: If a centroid has no points assigned, reinitialize it

Example (Demonstration)

kmeans_clustering.ipynb

- Demonstration of a custom implementation of the k-means algorithm with visualization of the learning process
- Several datasets and different initialization options

Questions:

- How does centroid initialization affect learning?
- How long does learning take for larger datasets?
- How to find the optimal number of clusters?
- How to evaluate the quality of the clusters formed?

The k-Means Clustering Algorithm

Advantages

- Fast algorithm, easy to implement
- Suitable for quick insight into data structure

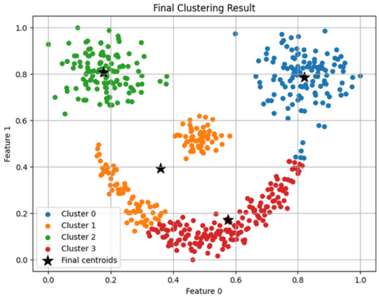
Disadvantages

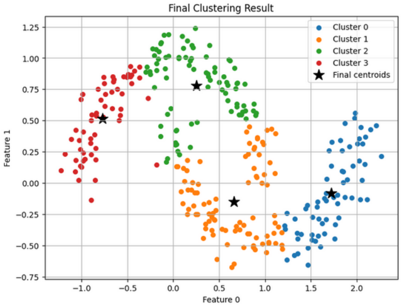
- The number of clusters must be specified in advance
- Batch processing (problematic for large data or online learning)
- High sensitivity to the initial choice of centroids
- Sensitive to outliers
- May fail for complex data structures: seeks spherical clusters
- Problematic for high-dimensional data (*curse of dimensionality*), or strongly correlated features

The k-Means Clustering Algorithm

Examples of More Complex Tasks:

kmeans_clustering.ipynb





More examples by ScikitLearn

The k-Means Clustering Algorithm

Disadvantages and Their Solutions

- The number of clusters must be specified in advance
  $\rightarrow$ try different values of k and choose the best one
- Batch processing (problematic for large datasets or online learning)
  $\rightarrow$ minibatch or online k-Means
- High sensitivity to the initial choice of centroids
  $\rightarrow$ enhanced initialization
- Sensitivity to outliers
  $\rightarrow$ data normalization:
  - also ensures invariance to scaling and translation
  - but may not always help (e.g., it may bring distant clusters closer)

The k-Means Clustering Algorithm

Disadvantages and Their Solutions

- May fail for complex data structures: tends to find spherical clusters
  $\rightarrow$ use a different distance metric
- Problems with high-dimensional input data (*curse of dimensionality*), or strongly correlated features
  $\rightarrow$ apply PCA (Principal Component Analysis) for input preprocessing

PCA Analysis

PCA (Principal Component Analysis)

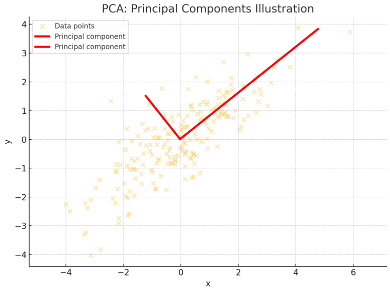
- Dimensionality reduction of input data:
  - use fewer features without losing essential information,
  - select the most important features (principal components).
- Each principal component (PC) is an orthogonal direction that captures the maximum possible variance in the data.
- The most important feature (the first principal component) is a unit vector $\vec{w}$ that maximizes the variance of the projections of all data points:

$$\vec{w} = \arg \max_{\|\vec{w}\|=1} \frac{1}{N} \sum_{i=1}^{N} (\vec{w}^\top \vec{x}_i)^2$$

- In other words, we seek the direction in which the data is the most "spread out".

PCA Analysis

PCA (Principal Component Analysis)



- Each principal component explains a portion of the total variance in the data.
- Common strategy: keep enough components to explain e.g. 90–95% of the total variance.

Metrics for Evaluating Clustering Quality

How can we tell if the clusters produced by an algorithm are actually good?

- Visual assessment works only for low-dimensional data (e.g., 2D or 3D)
- For general high-dimensional data, we need to define metrics that allow automatic evaluation:
  - **Compactness:** are the points within a cluster close to each other?
  - **Separability:** are the clusters well separated from each other?
- Such metrics can also be used to determine the optimal number of clusters
- *However, no single metric works best for all types of data and situations*

Metrics for Evaluating Clustering Quality

Silhouette

- Measures how close each point in a cluster is to points in the same cluster compared to points in other clusters.

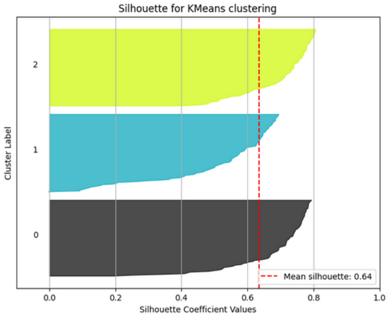
$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

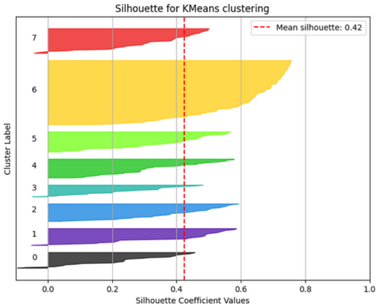
  - $S(i)$ is the silhouette score for point i
  - $a(i)$ is the average distance from point i to other points in the same cluster
  - $b(i)$ is the average distance from point i to points in the nearest neighboring cluster
- The closer the value is to 1, the better the point is assigned; values around 0 indicate boundary points; negative values suggest misclassification
- A popular metric for selecting the optimal number of clusters (it should be maximized)

Metrics for Evaluating Clustering Quality

Silhouette

- Results can be visualized using a **silhouette plot**:
  - each sample is represented by a horizontal bar (its length corresponds to $S(i)$)
  - ideally, all bars are long and positive
  - a red vertical line shows the average silhouette score across all points





Metrics for Evaluating Clustering Quality

Davies-Bouldin Index

- Measures the compactness of clusters and their separation
- Evaluates "outlierness" of clusters by comparing centroid distances and intra-cluster distances

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{s_i + s_j}{d(c_i, c_j)} \right)$$

- k is the number of clusters
- s_i is the average distance of the points in cluster i from its centroid c_i (compactness)
- $d(c_i, c_j)$ is the distance between the centroids of clusters i and j

Metrics for Evaluating Clustering Quality

Calinski-Harabasz Index

- Measures within-cluster similarity and between-cluster dissimilarity using variance

$$CH = \frac{B(k)}{W(k)} \times \frac{n-k}{k-1}$$

  - $B(k)$ is the between-cluster sum of squares
  - $W(k)$ is the within-cluster sum of squares
  - n is the total number of points

Metrics for Evaluating Clustering Quality

Within-Cluster Sum of Squares (WCSS)

- Sum of squared distances between individual points and the centers of their assigned clusters.
- Lower WCSS indicates more compact clusters.
- Typically decreases with increasing number of clusters k.

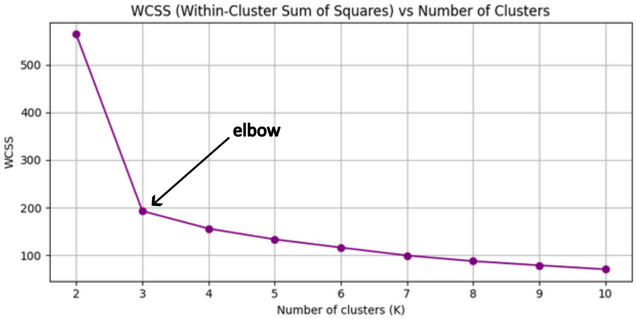
$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

  - C_i is the set of points assigned to the i-th cluster,
  - μ_i is the centroid of the i-th cluster.
- Used in the **Elbow Method** to determine the optimal number of clusters.

How to Use Metrics to Determine the Optimal Number of Clusters?

WCSS (Elbow Method):

- Monitor the decrease in within-cluster sum of squares.
- Choose k at the "elbow" point, where increasing k no longer significantly reduces WCSS.



How to Use Metrics to Determine the Optimal Number of Clusters?

- **WCSS (Elbow Method):**
  - Monitor the decrease in within-cluster sum of squares.
  - Choose k at the "elbow" point, where increasing k no longer significantly reduces WCSS.
- **Silhouette Score, Calinski-Harabasz Index:**
  - Select the k with the highest index value.
- **Davies-Bouldin Index:**
  - Select the k with the lowest index value (lower = better separation of clusters).

Note: Different metrics may suggest different optimal k — it is advisable to consider multiple criteria when making a decision.

kmeans_clustering.ipynb,

The k-Means Clustering Algorithm

Application: Vector Quantization

- The goal is to cover the input space as efficiently as possible using representatives, respecting the statistical distribution of the patterns
(similar to density estimation in statistics)
- Represent a set of vectors with a smaller subset of representative vectors
- Lossy data compression

vector_quantization.ipynb

- Explore how clustering can be used for vector quantization:
    - Try different numbers of centroids and compare the results
    - Experiment with different images (photographs)

Other Approaches to Clustering

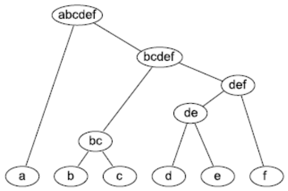
K-means is not the only way to partition data into clusters.

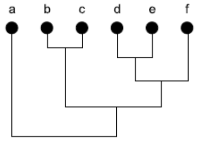
- Different algorithms are better suited for different types of data
- Some methods do not require specifying the number of clusters in advance
- Others can detect clusters of arbitrary shapes or handle categorical data
- Common alternatives include:
    - hierarchical clustering
    - density-based clustering (e.g., DBSCAN)
    - spectral clustering - uses the eigenvectors of a similarity graph to partition data into clusters
    - model-based clustering (e.g., Gaussian Mixture Models)

Nice comparison: Scikit Learn

Hierarchical Clustering

- No need to know the expected number of clusters in advance.
- Initially, each training sample represents its own cluster.
- A distance matrix between training samples is computed.
- During learning, the two closest clusters are merged iteratively.
- **Visualization:** dendrogram





Source: Kateřina Horaisová: Slides for the Neural Networks 2 course, FNSPE CTU Děčín

Hierarchical Clustering

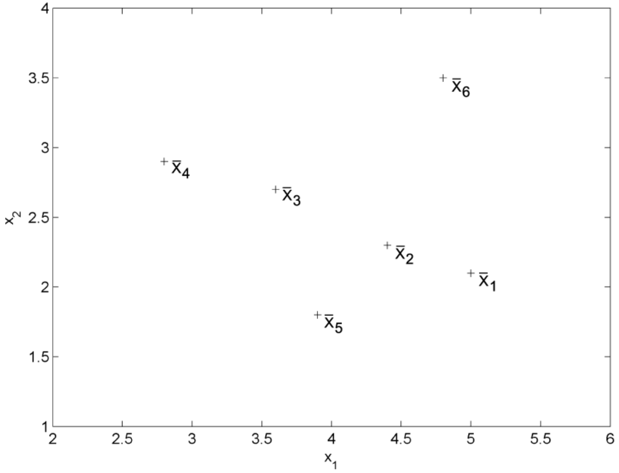
How to define distance between clusters C_i, $C_j \subset R^n$?

- **Single linkage (nearest neighbor)**
  $d(C_i, C_j) = \min\{d(\vec{x}, \vec{y}) | \vec{x} \in C_i, \vec{y} \in C_j, i \neq j\}$
- **Complete linkage (farthest neighbor)**
  $d(C_i, C_j) = \max\{d(\vec{x}, \vec{y}) | \vec{x} \in C_i, \vec{y} \in C_j, i \neq j\}$
- **Average linkage** $d(C_i, C_j) = \frac{1}{m_i m_j} \sum_{\vec{x} \in C_i} \sum_{\vec{x'} \in C_j} d(\vec{x}, \vec{y})$
- **Centroid linkage** $d(C_i, C_j) = d(\vec{\mu}_i, \vec{\mu}_j)$
- **Ward's minimum variance method**
  $d(C_i, C_j) = \frac{m_i m_j}{m_i + m_j} d(\vec{\mu}_i, \vec{\mu}_j)$

Hierarchical Clustering

- **Single linkage (nearest neighbor, min)**
  - Suitable for elongated clusters, but sensitive to noise.
- **Complete linkage (farthest neighbor, max)**
  - Prefers compact, spherical clusters.
  - Reduces the creation of elongated clusters.
- **Average linkage**
  - A compromise between single and complete linkage.
  - More stable in the presence of noise.
- **Centroid linkage**
  - Faster computation, but may sometimes incorrectly merge distant clusters.
- **Ward's method**
  - Minimizes the total within-cluster variance.
  - Produces compact and similarly sized clusters.

Hierarchical Clustering - Example



x_1	x_2
5,0	2,1
4,4	2,3
3,6	2,7
2,8	2,9
3,9	1,8
4,8	3,5

Source: Kateřina Horaisová: Slides for the Neural Networks 2 course, FNSPE CTU Děčín

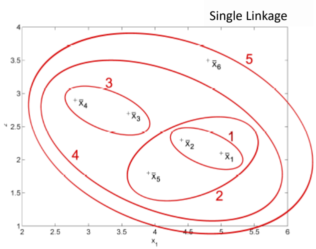
Hierarchical Clustering - Example

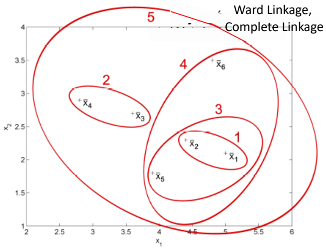
Distance matrix between training samples:

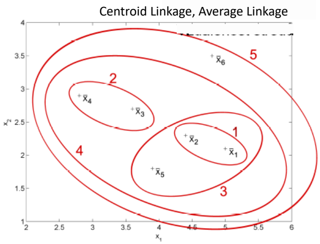
	$\overline{x}_1$	$\overline{x}_2$	$\overline{x}_3$	$\overline{x}_4$	$\overline{x}_5$	$\overline{x}_6$
$\overline{x}_1$	0,0000	0,6325	1,5232	2,3409	1,1402	1,4142
$\overline{x}_2$	0,6325	0,0000	0,8944	1,7088	0,7071	1,2649
$\overline{x}_3$	1,5232	0,8944	0,0000	0,8246	0,9487	1,4422
$\overline{x}_4$	2,3409	1,7088	0,8246	0,0000	1,5556	2,0881
$\overline{x}_5$	1,1402	0,7071	0,9487	1,5556	0,0000	1,9235
$\overline{x}_6$	1,4142	1,2649	1,4422	2,0881	1,9235	0,0000

Source: Kateřina Horaisová: Slides for the Neural Networks 2 course, FNSPE CTU Děčín

Hierarchical Clustering - Example

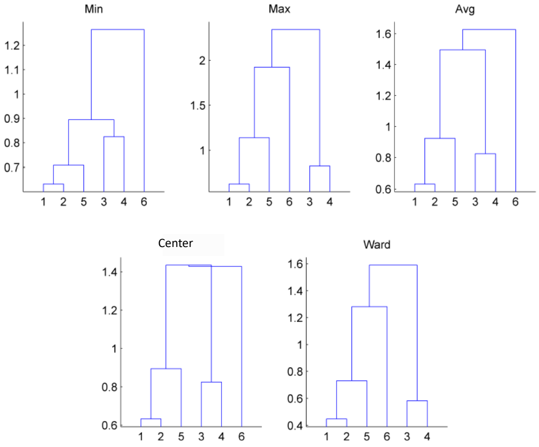






Source: Kateřina Horaisová: Slides for the Neural Networks 2 course, FNSPE CTU Děčín

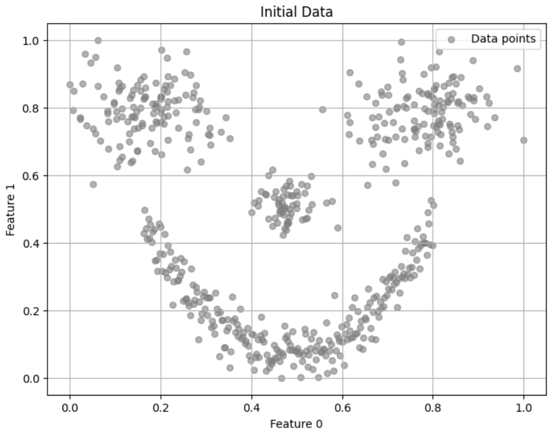
Hierarchical Clustering - Example



Source: Kateřina Horaisová: Slides for the Neural Networks 2 course, FNSPE CTU Děčín

Hierarchical Clustering -Example

Think about it: Which variant of hierarchical clustering would best handle the following example?



Hierarchical Clustering

Reminder: How to compute distance in clustering methods

- **Euclidean distance:** $d(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
- If only comparing distances (for efficiency):
  $d(\vec{p}, \vec{q}) = \sum_{i=1}^{n}(p_i - q_i)^2$

Alternative metrics:

- **Manhattan (city block) distance:** $d(\vec{p}, \vec{q}) = \sum_{i=1}^{n} |p_i - q_i|$
- **Chebyshev distance:** $d(\vec{p}, \vec{q}) = \max_i |p_i - q_i|$
- **Minkowski distance:** $d(\vec{p}, \vec{q}) = (\sum_{i=1}^{n} |p_i - q_i|^r)^{\frac{1}{r}}$
- **Cosine similarity:** $\cos(\vec{p}, \vec{q}) = \frac{\vec{p} \cdot \vec{q}}{\|\vec{p}\| \|\vec{q}\|}$
- (and others)

Example: Hierarchical Clustering

hierarchical_clustering.ipynb

- Demonstration of hierarchical clustering using the SciPy library
- Dendrograms shown for different linkage methods (single, complete, ward, etc.)

Questions:

- Observe differences between dendrograms produced using various distance metrics.
- Try to find the optimal combination of distance metric and number of clusters for each dataset.
- Can we determine the number of clusters from a dendrogram?

Example: Hierarchical Clustering

hierarchical_clustering.ipynb

Forming clusters based on a dendrogram:

- **Fixed number of clusters**
- **Cutting the tree at a distance threshold** – splits the dendrogram at a chosen height (height = linkage distance)
- **Adaptive cutting using inconsistency coefficient** – local decisions based on deviations from earlier merges

Inconsistency Coefficient in Hierarchical Clustering

The **inconsistency coefficient** helps identify where to cut the dendrogram based on how much a given merge deviates from previous (lower-level) merges.

$$\text{inconsistency}(i) = \frac{d_i - \mu_i}{\sigma_i}$$

- d_i – height (distance) of the merge at step i
- μ_i – average linkage height of the previous r levels
- σ_i – standard deviation of those r linkage heights

A higher value indicates that the merge likely connects structurally different clusters.

- We cut branches where the inconsistency exceeds a given threshold – this allows for cutting at varying heights across the tree.

Hierarchical Clustering

Advantages

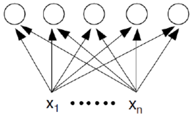
- Hierarchical structure, easy interpretation.
- No need to predefine the number of clusters, low parameter requirements.
- Handles clusters of different shapes better than k-means.

Disadvantages

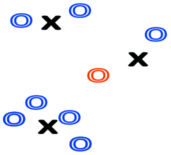
- Computationally expensive.
- Cannot update incrementally (online learning is not possible).
- Sensitive to the choice of distance metric and linkage method.
- Sensitive to outliers.
- Interpretation becomes difficult for a large number of clusters.

Competitive Models and Competitive Learning

- Let us return to the single-layer neural network model.



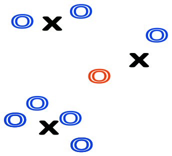
- **Key idea:** Neurons (representatives, agents) correspond to points in the input space, each representing one cluster (or part of it).
- The system of representatives (neurons) self-organizes in the input space (unsupervised learning = self-organization).



Competitive Models and Competitive Learning

Basic principle = competition:

- Neurons compete for the ,,right" to represent the given input pattern.



Learning goal:

- Place neurons in the centers of the pattern clusters (*centroids*) or in a way that reflects the data density (vector quantization).
- Preserve the network structure that has already been formed.

Competitive Models and Competitive Learning

K-means is actually a variant of competitive learning:

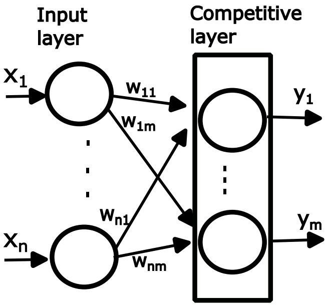
- **Winner-takes-all rule:** Centroids (representatives) compete to represent a training pattern. The nearest centroid wins and blocks others — it "takes all".

Typical mechanisms in competitive learning:

- **Winner-takes-all:** only the winning neuron is updated.
- **Not just winner-takes-all (soft competition):** the pattern affects not only the winner but also nearby neurons (neighbors).
- **Lateral inhibition:** the winner suppresses the activity of competitors.

Competitive Models – Basic Architecture

- A single-layer neural network.
- The input layer corresponds to the input features.
- The number of neurons in the output (**competitive**) layer corresponds to the (expected) number of clusters or representatives.



  - Each input neuron is connected to every output neuron.
  - There may also be **lateral connections** between output neurons.

Competitive Models – Basic Architecture

How do neurons compute their output (activation) for a given input pattern?

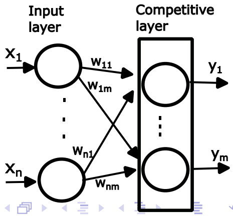
- As a distance between the presented input and their weight vector.
- For Euclidean distance (square root can be omitted):

$$d(\vec{x}, \vec{w}_i) = ||\vec{x} - \vec{w}_i||^2 = \sum_{j=1}^{n} (x_j - w_{ji})^2$$

- For cosine similarity (to be maximized):

$$d(\vec{x}, \vec{w}_i) = \cos(\vec{x}, \vec{w}_i) = \frac{\vec{x} \cdot \vec{w}_i}{||\vec{x}||\,||\vec{w}_i||}$$

For normalized vectors: $d(\vec{x}, \vec{w}_i) = \vec{x} \cdot \vec{w}_i$



Competitive Models – Principle of Competition

1. Present the input pattern $\vec{x}$
2. Neurons compute their activation as the distance between $\vec{x}$ and their weight vectors.
3. Neurons then compete for the right to represent the input.
4. The winning neuron (or several) updates its weights to move closer to the input.

Competition can be implemented in different ways:

1. via lateral connections and **lateral inhibition**
2. by directly comparing neuron activations (i.e., distances to the input)

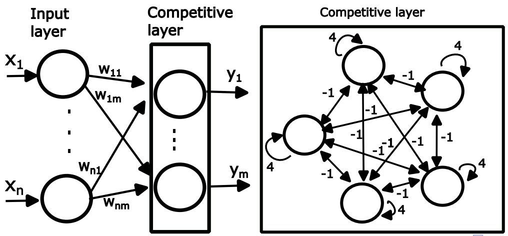
Competitive Models and Lateral Inhibition

Lateral inhibition principle:

- Output neurons are fully connected via fixed lateral weights: $t_{ii} = (k-1)$ and $t_{ij} = -1$ for $i \neq j$ (k = number of neurons)
- Each neuron iteratively updates its activation:

$$y_i = f\left(\sum_{l=1}^{k} t_{li} y_l\right)$$

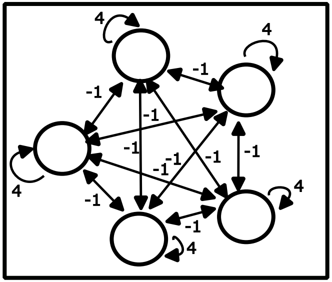
where f is typically a sigmoid function.



Competitive Models and Lateral Inhibition

- More active neurons suppress (inhibit) the activity of the others.
- **Expected result after a few iterations:** one neuron remains active, others are inhibited.

Competitive layer



Competitive Models and Lateral Inhibition

How to implement competition between neurons:

1. Using lateral connections and iterative computation:
    - Does not always produce good results – can blur the differences between neuron activities.
2. By comparing neuron outputs directly (i.e., distances to the input pattern):
    - The winner is the neuron with the highest activation (or lowest distance) – **winner takes all**.
    - Easier to implement, more stable in practice.

Competitive Models – Winner Takes All (WTA) Variant

Adaptation rule:

- The winning neuron i updates its weights to move closer to the presented input pattern $\vec{x}$:

$$\vec{w}_i(t+1) = \frac{\vec{w}_i(t) + \alpha\vec{x}}{\|\vec{w}_i(t) + \alpha\vec{x}\|}$$

(for cosine similarity, Hebbian rule)

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \alpha\left(\vec{x} - \vec{w}_i(t)\right)$$

(for Euclidean distance, difference rule)

Learning rate ... α

- $\alpha = 1$ ... complete update to match the input
- $0 < \alpha < 1$ ... partial update toward the input pattern
- $\alpha = 0$ ... no update (converged state)

With constant α, the network usually does not converge ... we require $\alpha \to 0$ over time.

Competitive Models – Winner Takes All (WTA) Variant Formal Algorithm (Euclidean Distance)

Input

- Training set $T = \{\vec{x}_1, ..., \vec{x}_N\}$ in $\mathbb{R}^n$
- Single-layer neural network with k output neurons.

Initialization

- Initialize weight vectors $\vec{w}_1, .., \vec{w}_k$ randomly (or by choosing k random examples from T)

Repeat:

- Randomly select an input $\vec{x} \in T$
- Compute $d(\vec{x}, \vec{w}_i)$ for $i = 1, ..., k$
- Choose winner neuron m such that $d(\vec{x}, \vec{w}_m) \leq d(\vec{x}, \vec{w}_i)$ for all i
- Update:

$$\vec{w}_m(t+1) = \vec{w}_m(t) + \alpha\,(\vec{x} - \vec{w}_m(t))$$

Competitive Models – Winner Takes All (WTA) Variant Applications

- **Clustering**
  - Online learning — essentially an **online version of the k-means algorithm** (i.e. k-means is the batch version of WTA competitive learning)
  - Easy identification of cluster representatives $\vec{w}_i$
  - The number of clusters is fixed in advance
  - More robust to noise than k-means
- Data compression, feature extraction, dimensionality reduction
- Anomaly detection

kmeans_clustering.ipynb
neural_gas.ipynb

Competitive Learning – Winner Takes All Variant Limitations

Common issues (similar to k-means):

- Learning rate α must be carefully chosen
- Weight initialization significantly impacts training speed
    - e.g. use randomly selected input vectors
- Distribution of neurons may not reflect data density
- **Dead neurons** – some neurons may never win
    - Normalize weight vectors
    - Controlled competition: track how often each neuron wins (conscience mechanism)
    - **Soft competition** (winner doesn't take all)
    - Introduce topological neighborhood structures, e.g., grid in a Kohonen layer

kmeans_clustering.ipynb
neural_gas.ipynb

Competitive Learning – "Winner Doesn't Take All" Variant

Neural Gas (soft competition):

- **Idea:** Each neuron has a degree of "sensitivity" to the training sample based on its distance from it.
- Unlike classic "winner-takes-all" where only one neuron is updated, here we update **all neurons** depending on how close they are to the input.
- Over time, both the learning rate and neighborhood size decrease — the system gradually stabilizes.

Variants:

- Sometimes, the neuron's rank (order by distance) is used instead of the distance itself.
- The winner moves toward the sample, and others in its vicinity may be pushed away.

Competitive Learning – “Winner Doesn’t Take All” Variant

Input:

- Training set $T = \{\vec{x}_1, ..., \vec{x}_N\}$ in $\mathbb{R}^n$
- Single-layer neural network with k output neurons

Initialization:

- Initialize weight vectors $\vec{w}_1, .., \vec{w}_k$ randomly (or by randomly selecting k samples from T).
- Set a large initial neighborhood and high learning rate.

Repeat:

- Randomly select $\vec{x} \in T$
- Compute $d(\vec{x}, \vec{w}_i)$ for $i = 1, ..., k$
- **Update:**
    - Adjust positions of all $\vec{w}_i$ based on their distance to $\vec{x}$ and the current neighborhood size
    - Decrease the learning rate and neighborhood size over time

Competitive Learning – “Winner Doesn’t Take All” Variant

Advantages of Neural Gas over WTA:

- **Smooth learning:** Multiple neurons are updated based on distance or rank.
- **More stable training:** Lower risk of so-called “dead neurons”.
- **Better space coverage:** Neurons tend to spread more evenly across the input space.

Disadvantages / limitations:

- Efficiency – the need to compute neuron rankings at every iteration
- Dynamic neighborhood – spatial relations among neurons can change during training

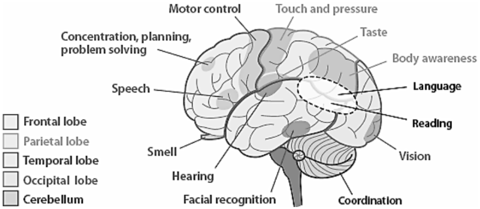
Example (Notebook):

neural_gas.ipynb

Self-Organizing Maps (SOM, Kohonen Maps) Teuvo Kohonen, 1981

- Original application: Phonetic typewriter (speech → text in Finnish)

Biological Motivation:

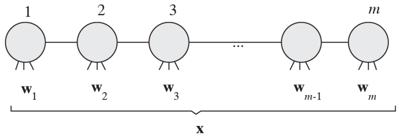


- In the cerebral cortex, specific areas of neurons are more responsive to certain types of stimuli.
- Physically nearby neurons tend to respond similarly – lateral connections cause excitation of nearby neurons and inhibition of more distant ones.

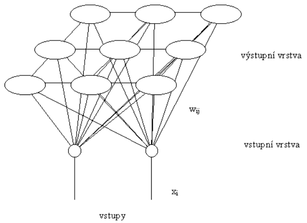
Source: `https://cybernetist.com/2017/01/13/self-organizing-maps-in-go/`

Self-Organizing Maps – Architecture (Topology)

1D architecture – linear chain:



2D architecture – grid:

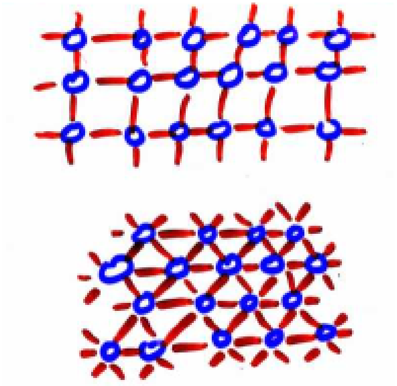


- Neurons are arranged topologically in a grid
- The grid defines **physical neighborhood** between neurons
  ($\times$ logical proximity in weight space)
- Neighboring neurons should respond to similar inputs

Paul Rojas: Neural Networks – A Systematic Introduction, Springer, 1996

Self-Organizing Maps – Grid Topologies

Examples of 2D grid topologies:

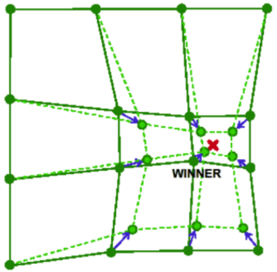


Self-Organizing Maps – Learning

Learning process:

1. Present an input vector $\vec{x}$
2. Each neuron computes its (Euclidean) distance to $\vec{x}$
3. The neuron with the smallest distance is the winner (Best Matching Unit – BMU)
4. The BMU and its neighbors update their weights
   - Neighboring neurons should respond similarly $\rightarrow$ mapping preserves topological structure

Weight adaptation:



SOM – Adaptation Rule

Adaptation

- Let k be the winning neuron for the presented input vector $\vec{x}$
- Each neuron i updates its weights according to the rule:

$$\Delta \vec{w}_i = \alpha \Lambda(i,k)(\vec{x} - \vec{w}_i)$$

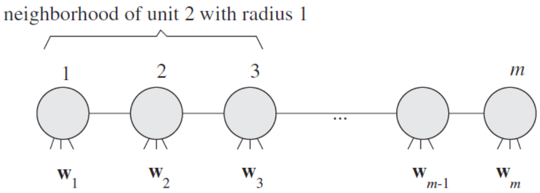
Neighborhood function = lateral interaction function $\Lambda(i,k)$

- Represents the strength of lateral interaction between neurons i and k during training
- Should decrease with increasing distance between neurons i and k in the grid
- Determines **whether and how strongly neuron i will be updated**, depending on its distance from the winning neuron k

SOM – Neighborhood Definition (1D Chain)

1D Neighborhood:

- Neurons are arranged in a sequence and indexed $1, \ldots, m$
- A neuron with index k has neighbors $k-1$ and $k+1$ (unless on the edge)

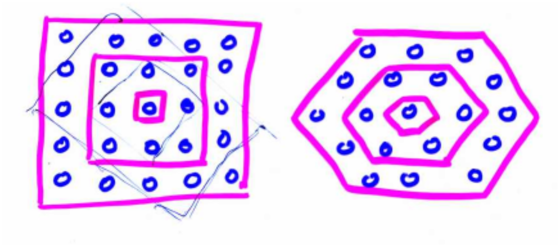


Paul Rojas: Neural Networks – A Systematic Introduction, Springer, 1996

SOM – Neighborhood Definition (2D Grid)

Neighborhood in multiple dimensions (2D grid):

- Similarly, the neighborhood of a neuron k (radius $= 1$) includes neurons connected via lateral links
- Any grid metric can be used: square, hexagonal, etc.



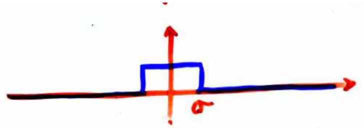
SOM – Neighborhood Function (Lateral Interaction)

Neighborhood function $\Lambda(i,k)$**:**

- $\Lambda(i,k)$ = strength of lateral connection between neurons i and k during learning
- Should decrease with increasing distance between neurons i and k

Example – Discrete Neighborhood:

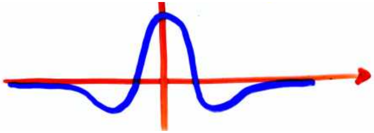
- $\Lambda(i,k) = 1$ if neuron i is in neighborhood of k (within radius σ), otherwise 0
- Efficient to implement – only neighbors are updated
- σ = neighborhood width (e.g. $\sigma = 1$)



SOM – Neighborhood Function (continued)

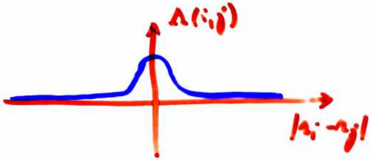
Mexican Hat function:

- Most biologically realistic



Gaussian function:

- $\Lambda(i,k) = \exp\left(-\frac{|\vec{w}_i - \vec{w}_k|^2}{\sigma^2}\right)$
- σ is the neighborhood width (typically decreases: $\sigma \to 0$)



SOM – Weight Adaptation and Parameters

Weight Update:

- Let k be the winning neuron (BMU) for input $\vec{x}$
- Each neuron i updates its weights as:

$$\Delta \vec{w}_i = \alpha \Lambda(i, k)(\vec{x} - \vec{w}_i)$$

Adjustable Parameters:

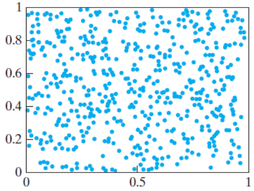
- **Learning rate** α (vigilance coefficient): $\alpha \in (0, 1)$
  - Fixed α prevents convergence – typically $\alpha \to 0$
- **Neighborhood width** σ
  - Typically decreases: $\sigma \to 0$

SOM – Learning Algorithm

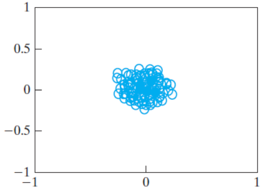
1. **Initialization:**
   Randomly initialize weights of output neurons. Set initial learning rate α, neighborhood width σ, and interaction function Λ.
2. **Repeat:**
   1. Present the next training vector $\vec{x}$
   2. Compute distance d_i between $\vec{x}$ and $\vec{w}_i$ for each output neuron:
      $$d_i = \sum_j (x_j - w_{ji})^2$$
   3. Select neuron k with smallest d_k as the winner
   4. Update weights of all (or neighboring) neurons:
      $$\vec{w}_i(t+1) = \vec{w}_i(t) + \alpha(t)\Lambda(i,k)(\vec{x} - \vec{w}_i(t))$$

Self-Organizing Maps

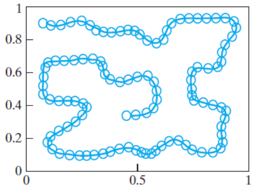
Example – Uniformly distributed data and 1D chain



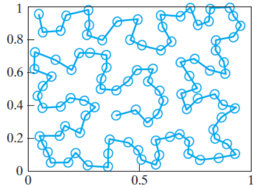
(a) Input distribution



Time = 0
(b) Initial weights



Time = 50 K
(c) Ordering phase

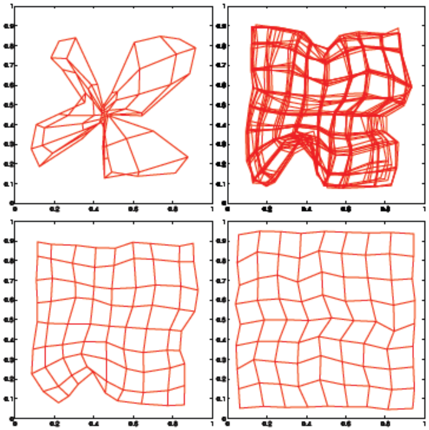


Time = 100 K
(d) Converging phase

Simon Haykin: Neural Networks and Learning Machines, 3rd edition, 2008

Self-Organizing Maps

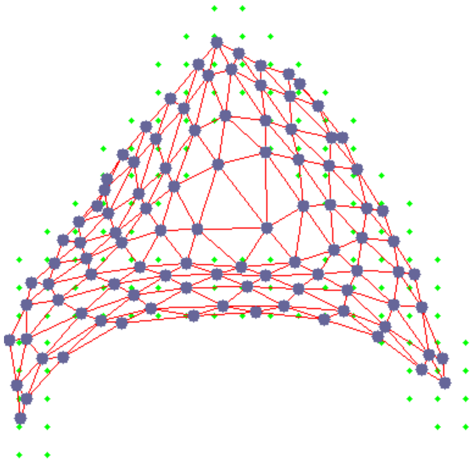
Example – Uniformly distributed data and 2D grid



Paul Rojas: Neural Networks – A Systematic Introduction, Springer, 1996

Self-Organizing Maps

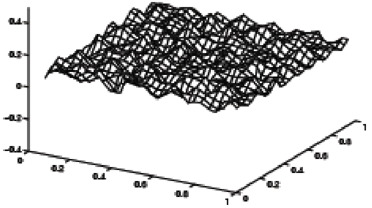
Example – Non-uniformly distributed data and 2D grid

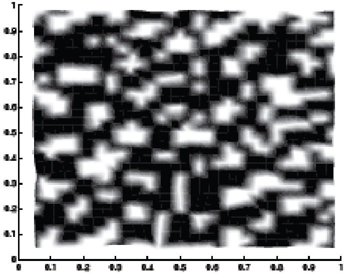


Self-Organizing Maps – Interpretations

Two possible interpretations for applications:

1. Dimensionality reduction with topology preservation
2. Clustering



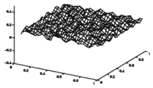
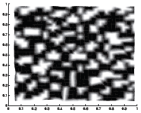


SOM – Dimensionality Reduction with Topology Preservation

- The network maps an n-dimensional input space into a 2D output space
- Neighborhood in the input space is preserved
- For very dense grids, the transformation is continuous
- Enables effective data visualization

Example:

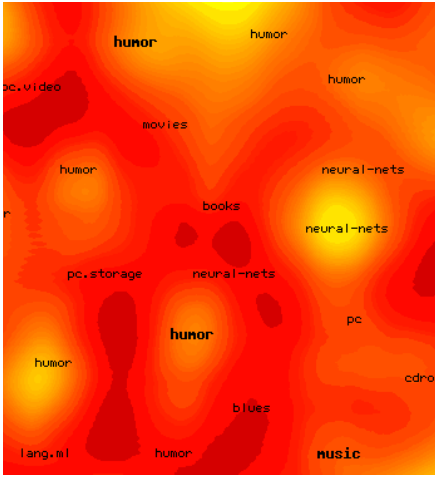
- 2D grid in 3D input space

SOM – Dimensionality Reduction Example: WebSOM

Dimensionality reduction with topology preservation

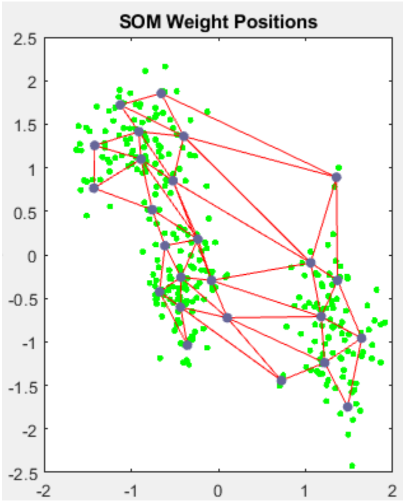
- Application example: WebSOM (1998)
- 2D visualization of similarity among web documents



SOM – Clustering Interpretation

Clustering interpretation:

- The weight vector of each output neuron represents a point in the input space
- Neurons cover the input space and reflect its density (vector quantization)
- Output neurons serve as cluster representatives
- Additionally, the neighborhood structure is preserved



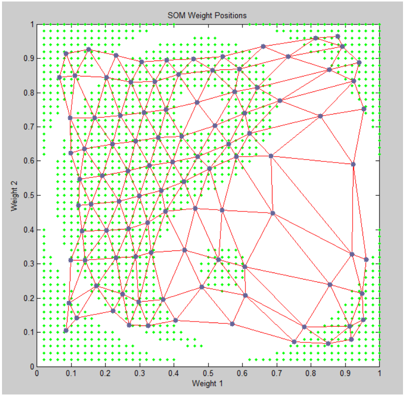
SOM – Learning Algorithm Analysis

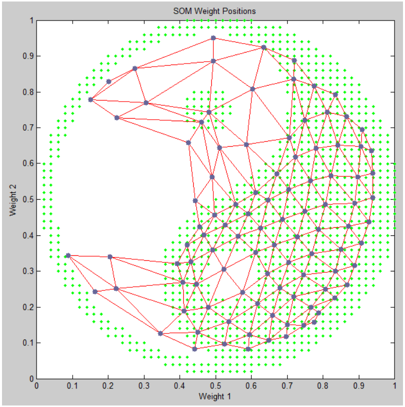
Key questions: How well has the SOM learned?

- Has the algorithm converged? How long does training take?
- To what extent is the topology preserved?
- Is the resulting mapping correct or meaningful?
- How do hyperparameters α, σ influence the result?

SOM – How well has the network learned?

Example: a well-trained model vs. a suboptimal one





SOM – How well has the network learned?

Within-Cluster Sum of Squares (WCSS):

- Measures compactness of clusters: sum of squared distances from inputs to their best-matching unit (BMU)

$$WCSS = \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} \|\vec{x} - \mu_i\|^2$$

Topographic Error:

- Measures topology preservation: proportion of samples where the first and second BMUs are not adjacent in the SOM grid

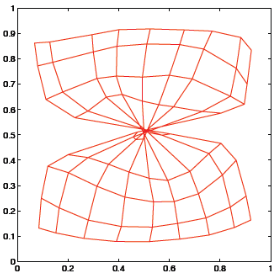
$$E = \frac{1}{n} \sum_{i=1}^{n} u(\vec{x}_i)$$

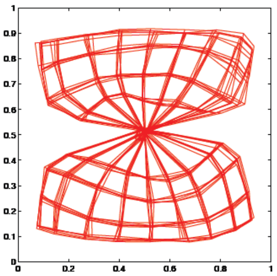
where $u(\vec{x}_i) = 1$ if the first and second BMUs are not neighbors, otherwise 0.

SOM – Impact of Parameters

Choice of parameters α (learning rate), σ (neighborhood width) has a major impact:

- Depends on the task and dataset
- Rapid decrease of σ can lead to topological defects (e.g., map twisting or folding)





- Rapid decrease of α can freeze the learning process in a poor local minimum or prevent convergence entirely

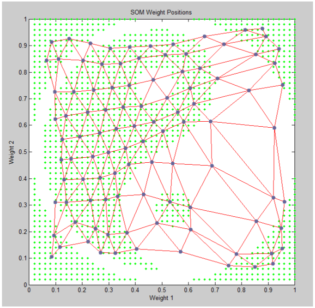
SOM – Two-Phase Adaptation Strategy

Two distinct learning phases:

- **Organization phase (topology shaping):**
  - Neighborhood covers most or all of the map
  - α is relatively large and stable
- **Convergence phase (fine-tuning):**
  - Neighborhood size shrinks to 1 (only the BMU is adapted)
  - α decreases rapidly toward 0

SOM – Visualization

2D input data – easily visualized:



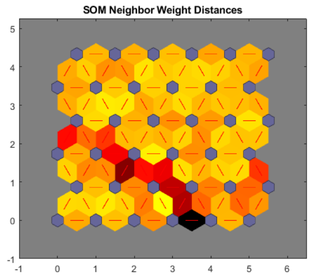
For higher dimensions:

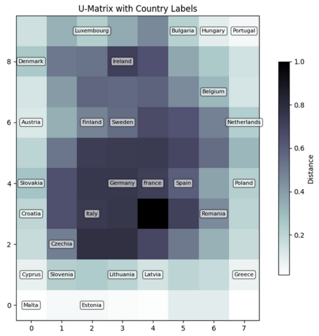
- **U-matrix (Unified Distance Matrix)** – shows distances between neighboring neurons, **Weight Planes**
- **Projection into 2D:**
  - Using PCA
  - Using Sammon's mapping

SOM – U-Matrix Visualization

U-Matrix (Unified Distance Matrix):

- Matrix showing distances between neighboring neurons' weight vectors
- Darker color = larger distance
- Clusters appear as "valleys", boundaries as "ridges"

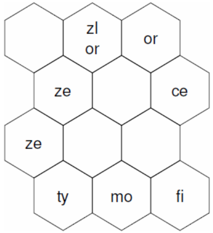


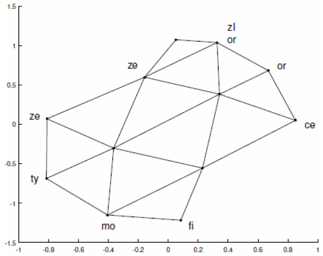


SOM – Sammon Mapping

Sammon Projection:

- Repositions data points in a low-dimensional space (not axis projection)
- Attempts to preserve distances between data points





Source: Kateřina Horaisová – Neural Networks 2 (FJFI ČVUT Děčín)

Kohonen Maps – Examples

- We will work with the **MiniSom** library
- **SOM_countries.ipynb** - dimensionality reduction
- **SOM_clustering.ipynb** - clustering