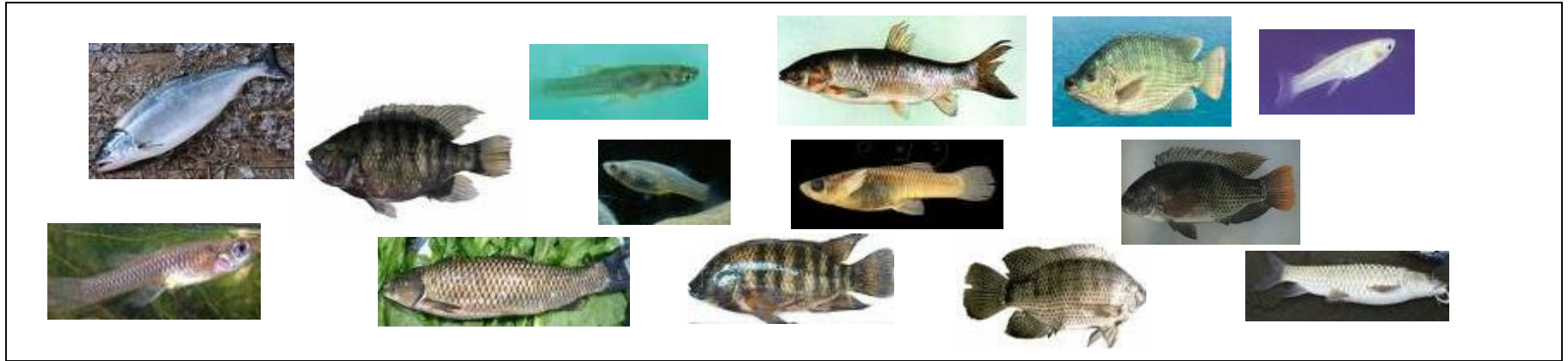


# Unsupervised Learning

# Unsupervised Learning

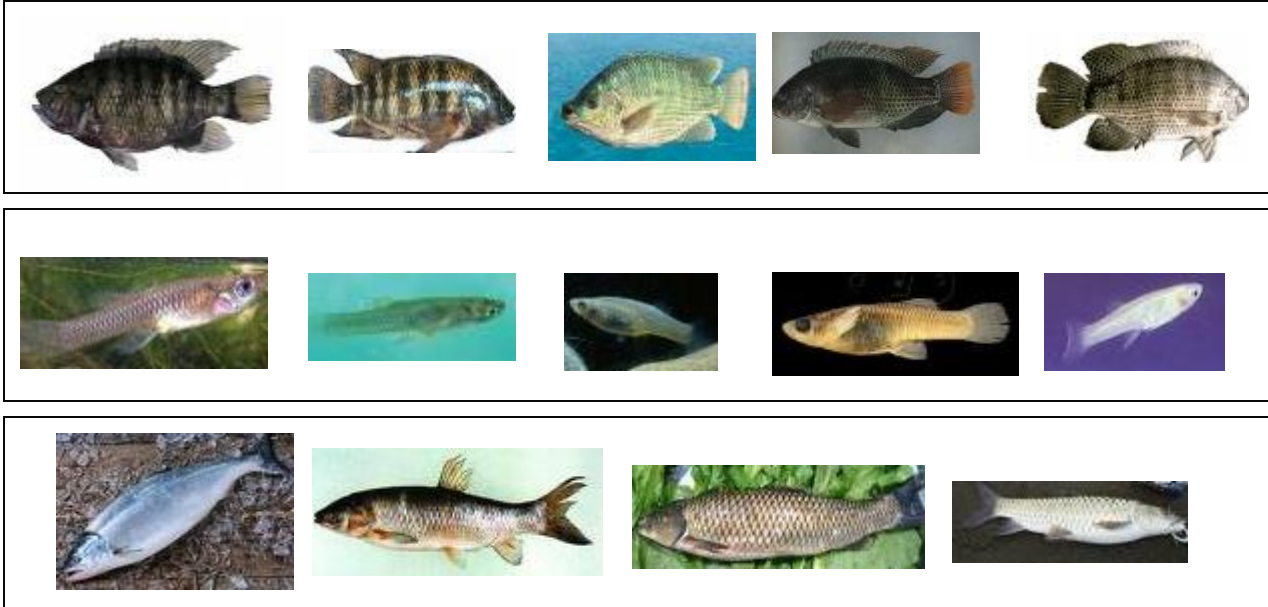
Data with **NO** labels:



- We do not know what the classes are.
- To group these sample images into clusters, we need to consider:
  - What attributes to use?
  - How to form the clusters?

# Unsupervised Learning

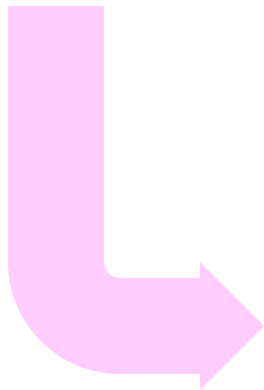
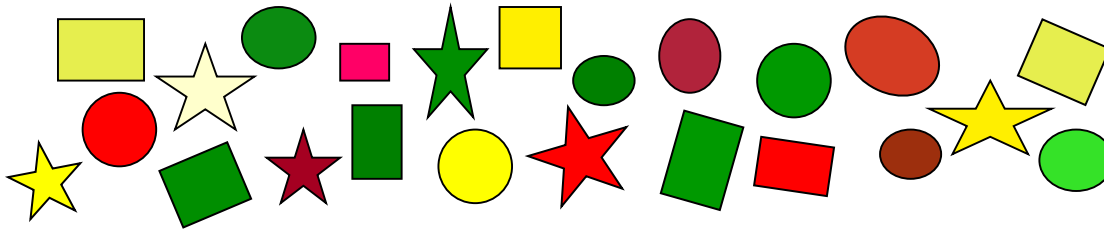
Example clusters:



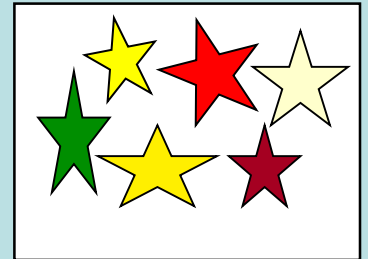
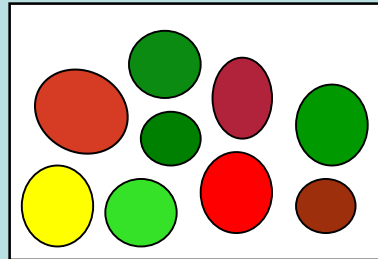
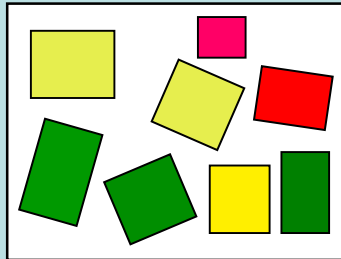
In clustering, there are no "correct" or "incorrect" results. We can only look for "reasonable" or "informative" results based on what we know about the data.

# Clustering Criterion

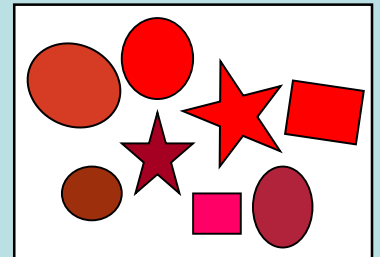
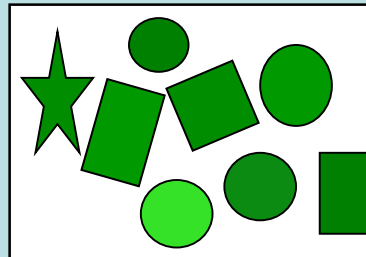
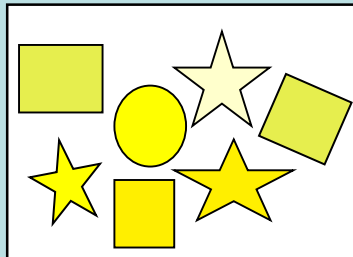
Clustering criterion is what we want to base the clustering on. The exact choice depends on the algorithm and the type of clusters desired.



By  
shape:



By  
color:



# Clustering as Partition of Data

Let  $C_i$  be the  $i^{\text{th}}$  cluster of our data set  $X$ . it can be considered a set containing all the feature vectors in it. Let there be a total of  $m$  clusters.

$$C_i \neq \phi, \quad \forall i$$

$$C_1 \cup C_2 \cup \dots \cup C_m = X$$

The following property makes this a hard (or crisp) partition (i.e., each  $x$  belongs to exactly one cluster):

$$C_i \cap C_j = \phi, \quad \forall i \neq j$$

The set of all the clusters for a dataset is usually called a **clustering** or a **partition** of the data:

$$\mathcal{R} = \{ C_1, C_2, \dots, C_m \}$$

# Proximity Measure

- Since our goal is to group "similar" patterns into clusters, we need a measure of how similar (or conversely, dissimilar) two patterns are.
- The choice of proximity measure directly affects the "shape" of clusters in the feature space.
- Proximity between two clusters can be computed from the proximity between their members.

# Cluster Representatives

It is often useful to represent a cluster by a few parameters (instead of a list of all its members).

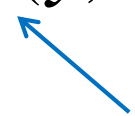
The most common form is a point representative. Examples:

mean point  $m_p$ : 
$$m_p = \frac{1}{n_C} \sum_{y \in C} y$$

medoid  $m_c$  (one of the vectors in the cluster):

$$m_c = \arg \max_{z \in C} \sum_{y \in C} s(y, z)$$

similarity measure



# k-Means

- The most popular and widely used clustering algorithm.
- An example of **competitive learning**, a main type of unsupervised learning methods.
- The number of clusters is predetermined (the "*k*").
- Cluster representatives (prototypes) required. The standard form uses “mean points” (the “*means*”).
- There are many variations:
  - Non-point prototypes
  - Partially overlapping clusters (soft partitions)
  - etc.



# k-Means

Each sample belongs to exactly one cluster

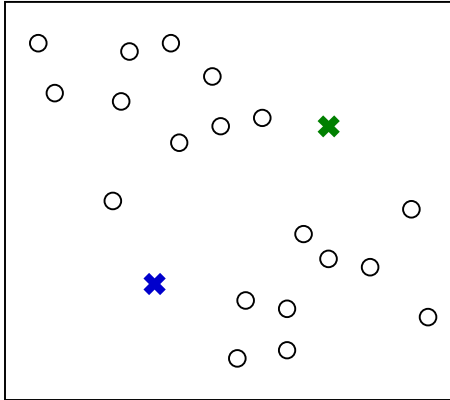
Algorithm:

- (Randomly) initialize the prototypes
- Repeat until "stopping criteria"
  - Assign each sample to the cluster of the closest prototype (the *competitive* part)
  - Recalculate each prototype as the mean of all  $x$  belonging to that cluster

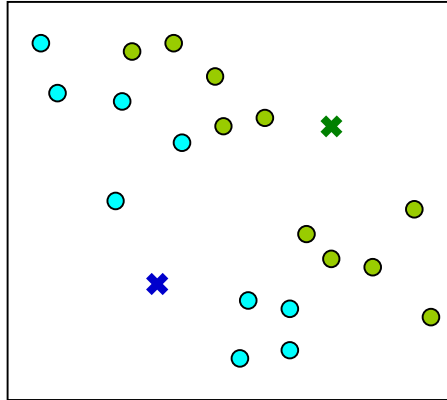
Common stopping criterion: no more change of prototypes (or cluster assignments) between iterations.

# k-Means Example

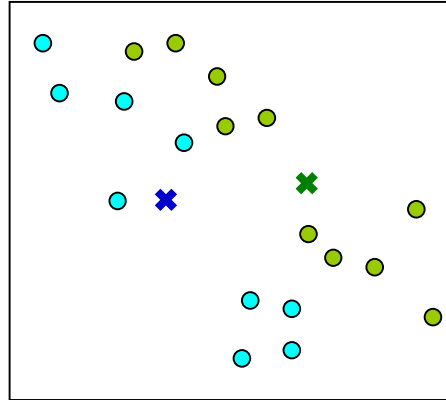
Initialization



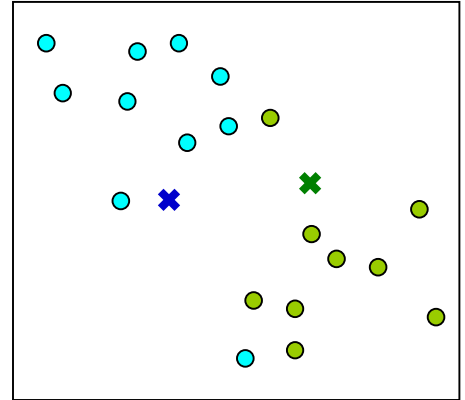
Assign Clusters



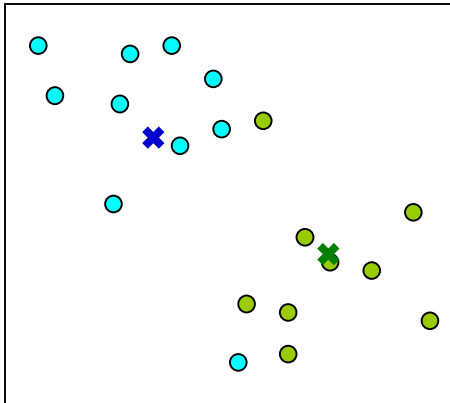
Update Prototypes



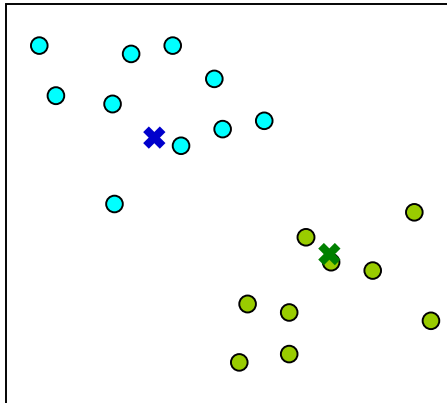
Assign Clusters



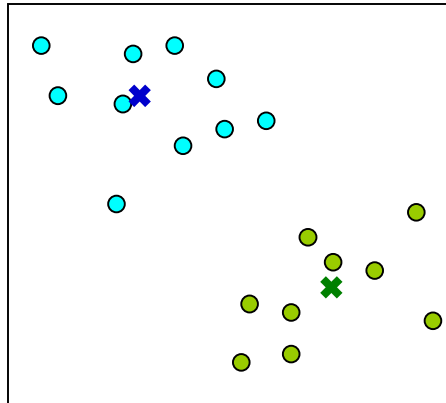
Update Prototypes



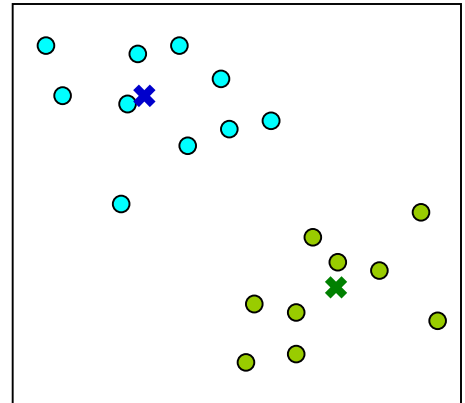
Assign Clusters



Update Prototypes



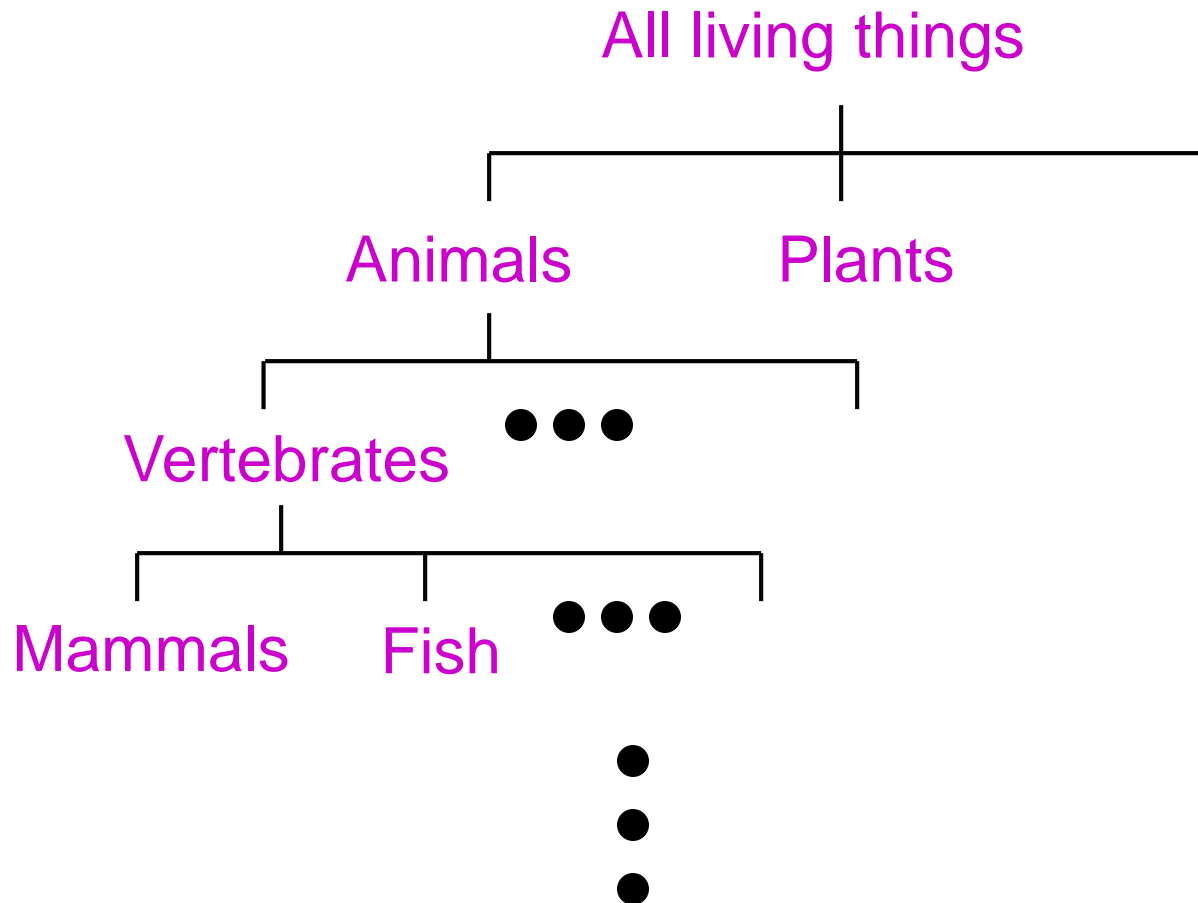
Assign Clusters



No change → Stop!

# Hierarchy Clustering

Hierarchical clustering algorithms create a hierarchical organization of all the samples. Each cut (horizontally) gives a different partition of the data.



# Relational Data Clustering

Hierarchical clustering is the most common approach for clustering **relational data**:

- The only information used is the "relations" (similarities or distances) among the samples.
- No need to represent the samples with vectors of attributes.
- Relations represented as a matrix or a graph.
- Particularly useful in applications where it is difficult or meaningless to use factored representation. Examples:
  - Biology (see the previous slide)
  - Document, music, etc.
  - Network structure (e.g., social network)

# Hierarchical Algorithms

Two main types of hierarchical clustering algorithms:

- Agglomerative Clustering Algorithms:

- Initial state: One cluster for each sample

- Each iteration: A pair of clusters are merged

- Divisive Clustering Algorithms:

- Initial state: One single cluster containing all the samples

- Each iteration: One of the clusters is divided into two

# Some More Clustering Approaches

There are a large number of different approaches used in clustering. Several major categories are:

- Mode-seeking (e.g., mean-shift): Iterative update of cluster representatives so that they move toward local maximums of data density.
- Mixture decomposition: Approximate the data distribution with the weighted combination of several parametric distributions.
- Density-based (e.g., DBSCAN): Use local data density for the decision of whether to put samples in the same cluster.
- Graph-cut (e.g., spectral clustering): Represent the similarities among the samples as a weighted graph, then find a “cut” of the graph.
- Many more ...

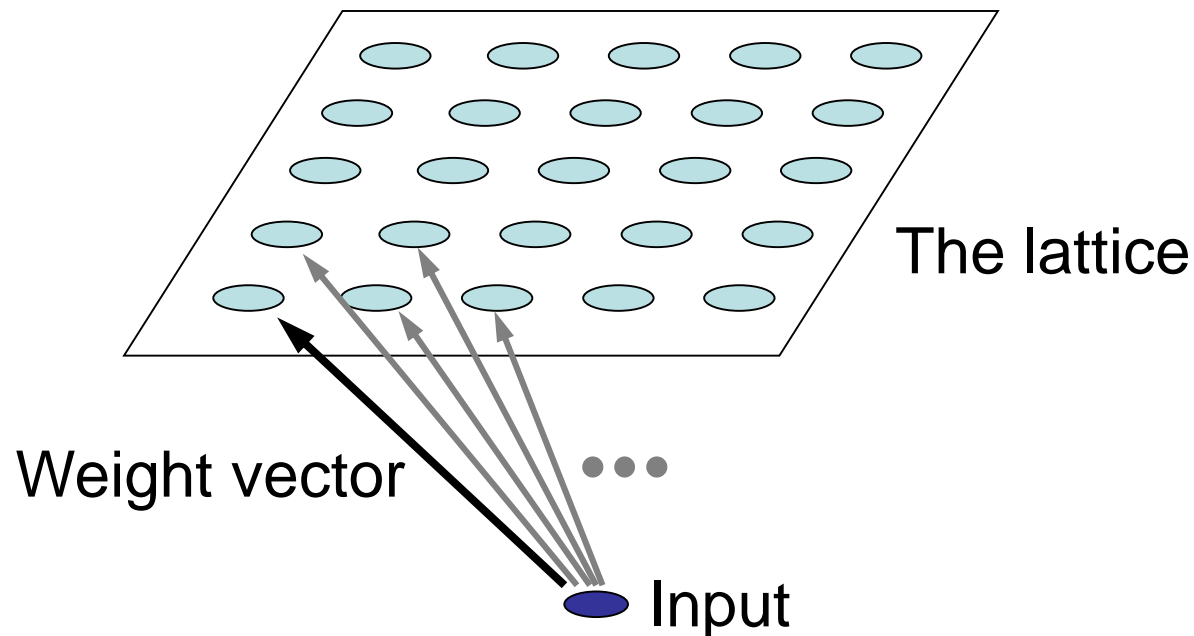
# Vector Quantization (VQ)

This is a really brief introduction:

- A fixed number of randomly initialized vector representatives.
- For each step:
  - Select an input sample.
  - Move the nearest representative towards it (with a learning rate).
- Repeat until convergence.
- Very similar to k-means (it's an online version) ...
- The most common application is lossy compression. (It's a part of many earlier codecs.)

# Self-Organizing Maps (SOM)

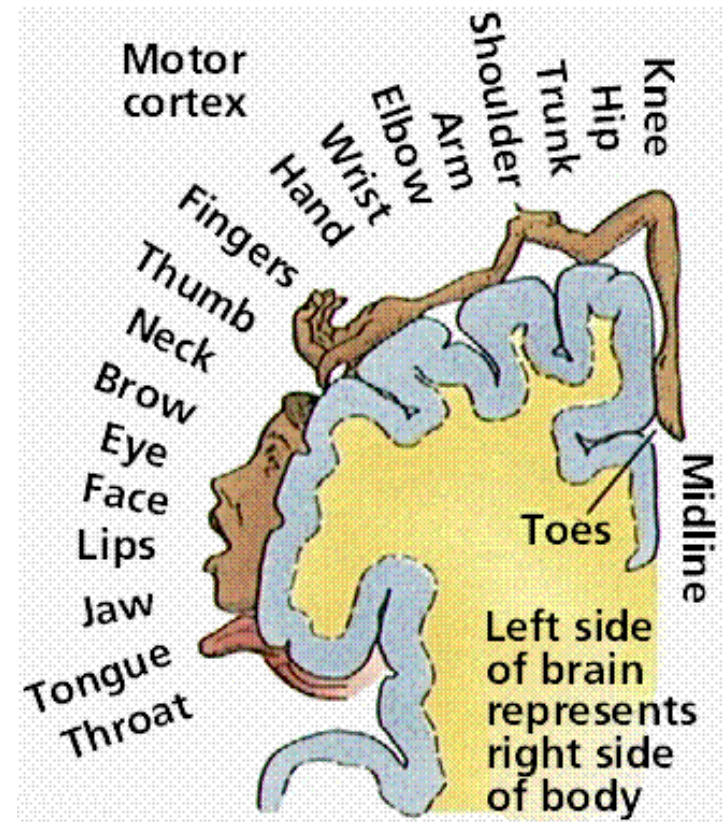
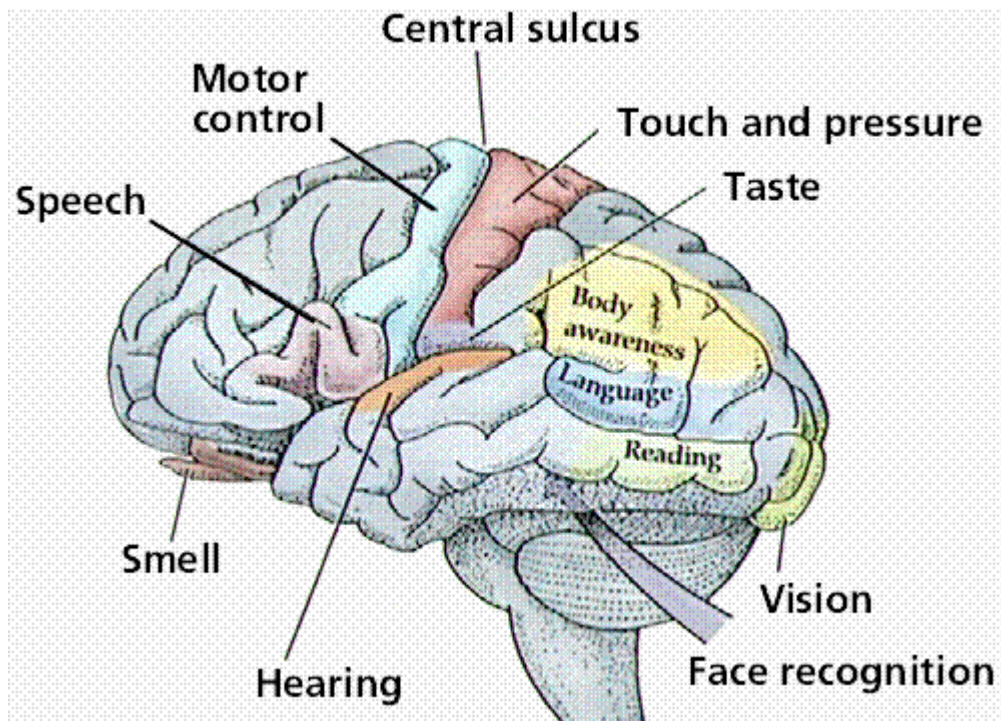
- A SOM has a single layer of neurons arranged on a lattice (2-D is most common).
- Nearby neurons on the lattice should behave similarly (have similar responses to inputs). Overall, the resulting lattice gives an organized representation of the inputs.





# Biological Origin of SOM

SOM is inspired by how the information processing in a human brain is distributed and organized at the cerebral cortex:



# Training a SOM

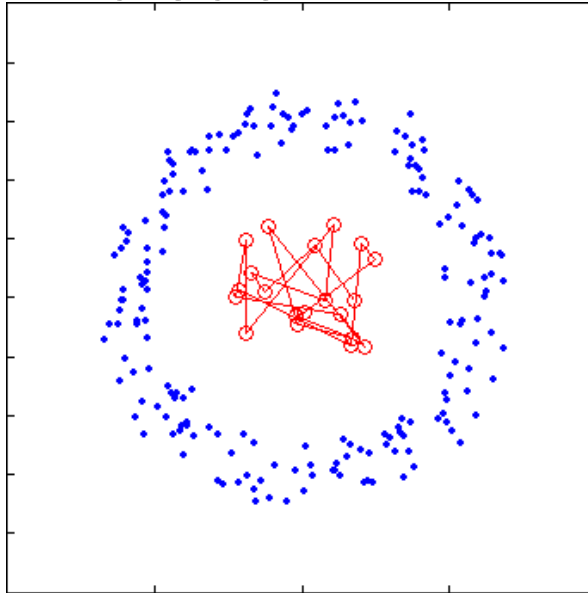
- A typical representation of SOM neurons:
  - The "weights" of a neuron is a point in the feature space.
  - The response of a neuron to an input is based on their proximity in the feature space (the closer they are, the stronger the response).

# Training a SOM

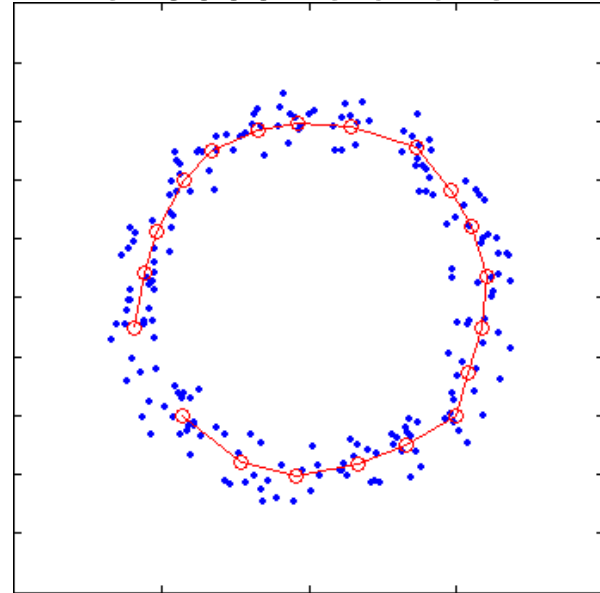
- Random initialization of the weight vectors.
- During training, for each input sample:
  - Competition: Select the winning neuron as the one that generates the strongest response.
  - Adaptation: The weight vector of the winning neuron is moved toward the input (the amount is affected by the learning rate) so that it will respond stronger to the same input next time.
  - Cooperation: The weight vectors of other neurons are moved as well, with the amount depending on their distances to the winning neuron. This is to ensure that nearby neurons will respond to inputs similarly.
- Repeat the above until convergence.

# SOM Example

Initial state

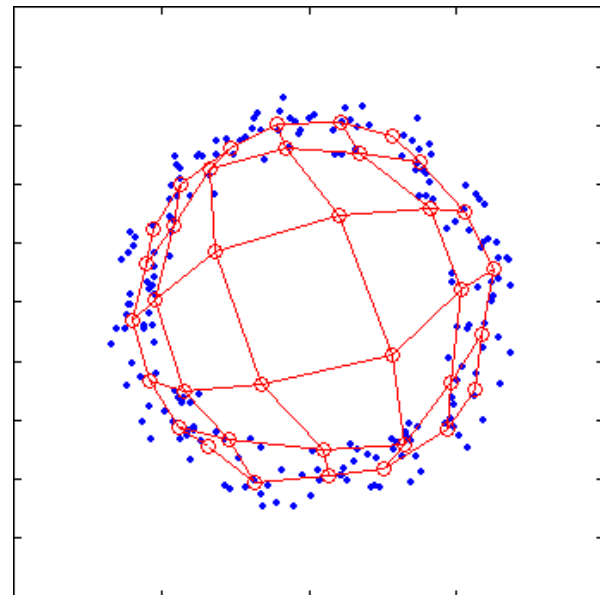
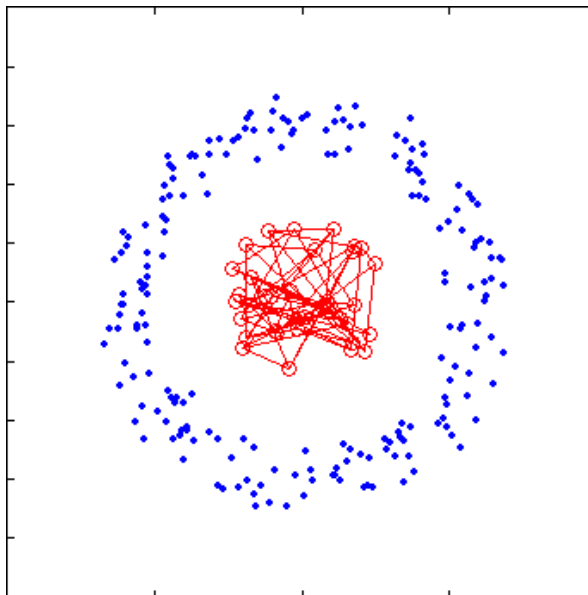


After 5000 iterations

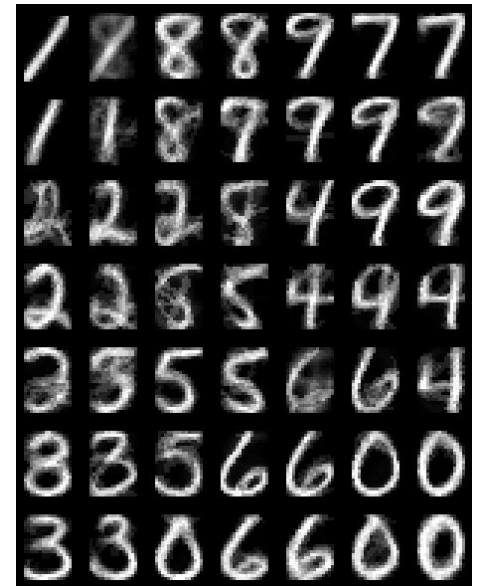
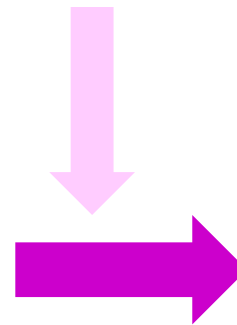
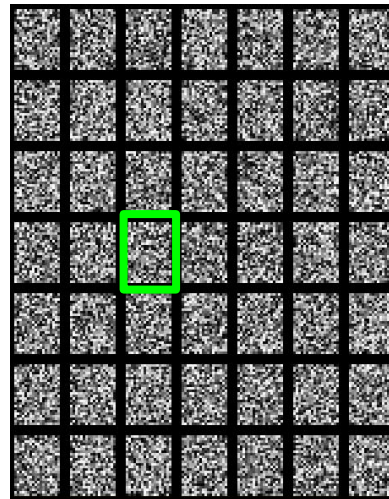
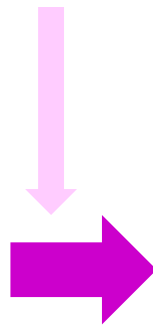
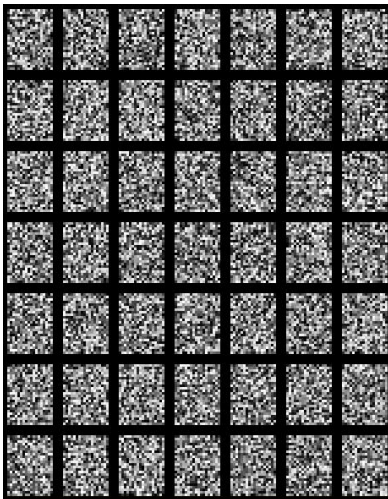
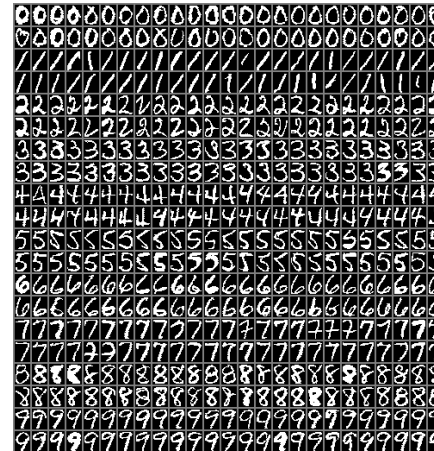


1-D SOM  
(20 nodes)

2-D SOM  
(6x6)



# SOM Example



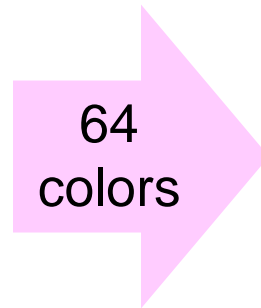
# Applications of SOM

- A reduced-dimension representation of the original data.
- The distribution of neuron weight vectors approximates the "density" of the training data.
- The approximate topology (proximity) is preserved. (This is how SOM differs from methods like VQ.)
- Many tasks (such as training a classifier) can be done on this lower dimensional representation. As a result, SOM is often called **SOFM** (self-organizing feature map) as well.
- Visualization of high-dimensional data.



# Applications of SOM: Example

- SOM applied to the RGB values of image pixels.



8x8  
SOFM



32x32  
SOFM



# Dimensionality Reduction

- Comparison: Standard SOM and VQ attempts to map an input vector (a sample) to the most similar representative, which is also in the input space.
- A very important category of techniques for reducing the total amount of data is to find a representation in a space of lower dimension. Such techniques are called **dimensionality reduction**.
  - Dimensionality reduction leads to a lossy representation, so it's important to preserve as much information useful for the task as possible.

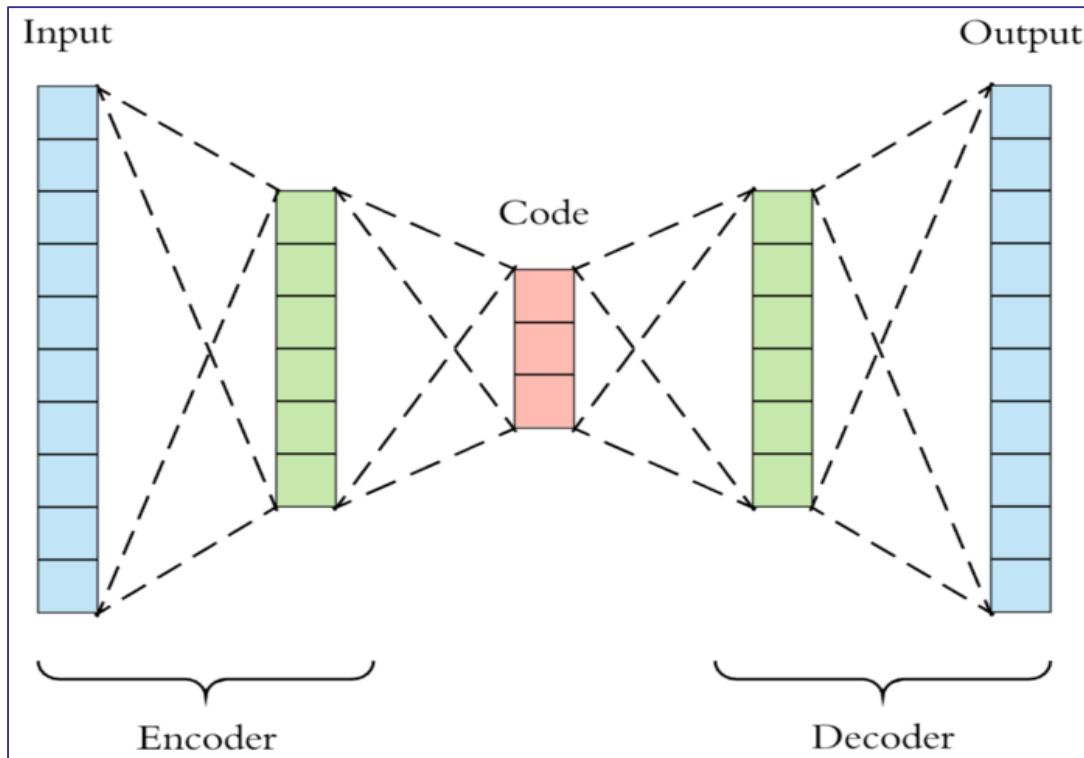


# Dimensionality Reduction

- Linear methods of dimensionality reduction aims to find a projection onto a subspace of the input space.
- Two very common linear dimensionality reduction methods:
  - **Principle Component Analysis (PCA)**: The aim is to preserve the amount of variance of the inputs.
  - **Linear Discriminant Analysis (LDA)**: The aim is to preserve the separability among inputs of different classes.
  - They are not considered “learning” based methods, so we will not get into their details. However, they are fundamental techniques for data analysis.

# Autoencoder

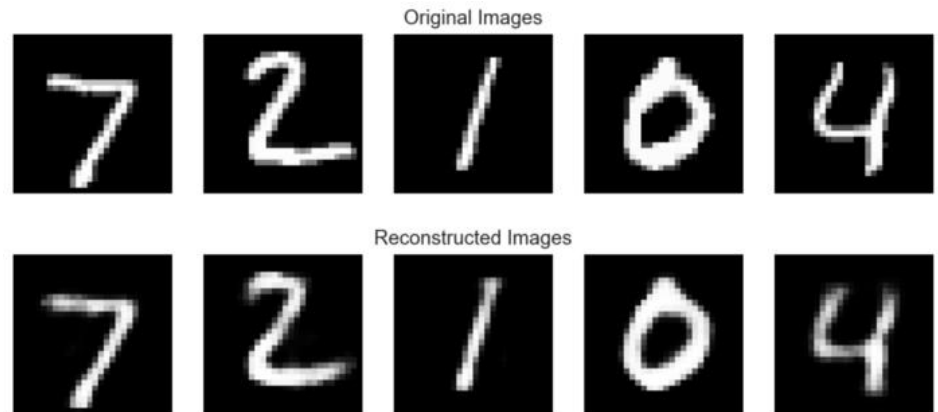
- **Autoencoder** is a learning based technique for dimensionality reduction.
- The objective is for the output to “reproduce” the input.
- The “information bottleneck” leads to dimensionality reduction, forcing the network to learn only the “important” information from the training samples.



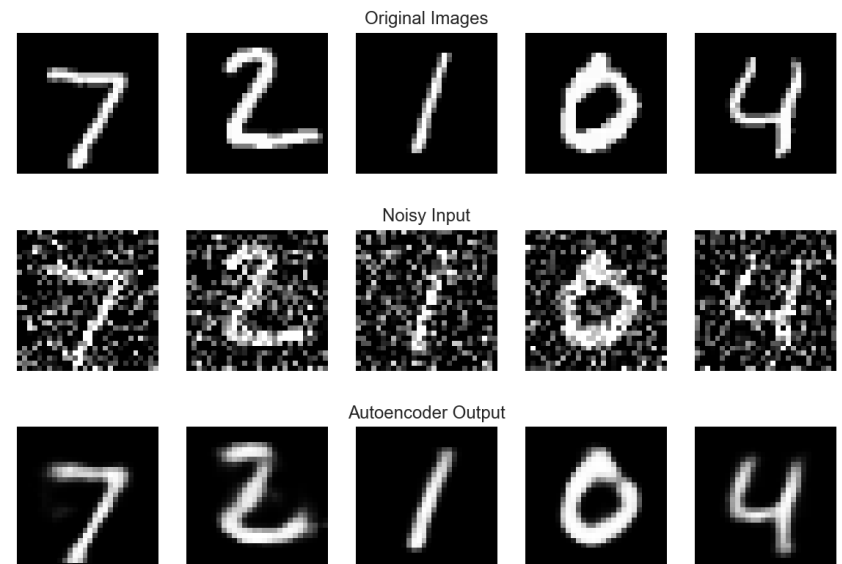
<https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798>

# Autoencoder Examples

- Reconstruction from a 32-element code.  
(Original input space has 784 dimensions.)



- Image denoising.  
(Trained with noisy-clean image pairs.)



- Many other applications ...