INF3490 unsupervised learning

Kyrre Glette
Slides mostly from Arjun Chandra

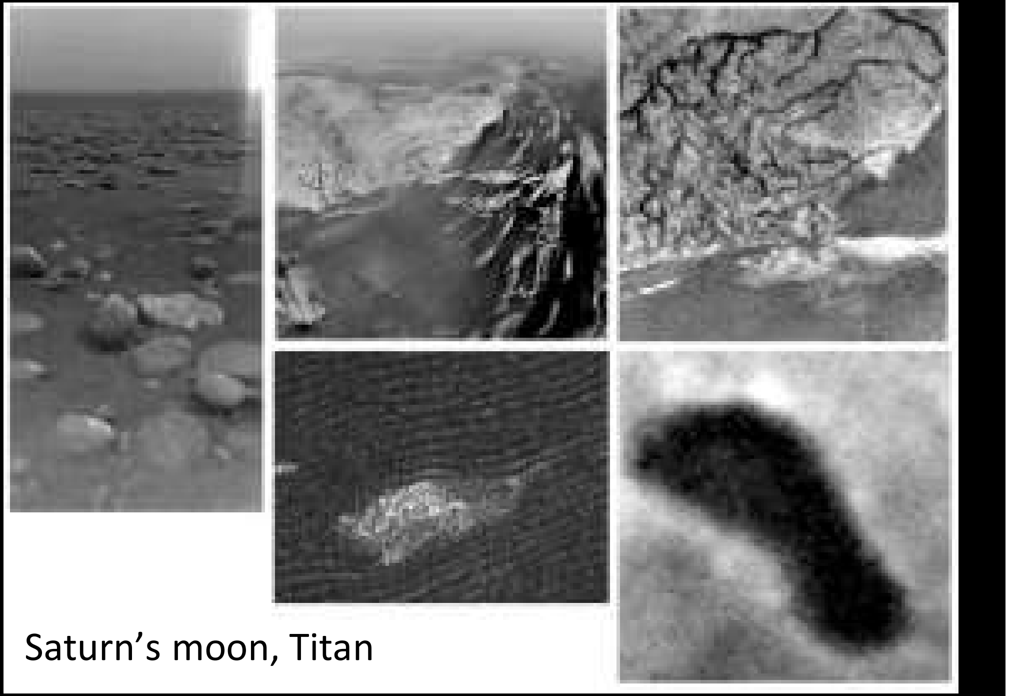
supervised learning?

- training data is **labelled (targets provided)**
- targets used as **feedback** by the algorithm to **guide learning**

what if there is data but **no targets**?

unknown targets

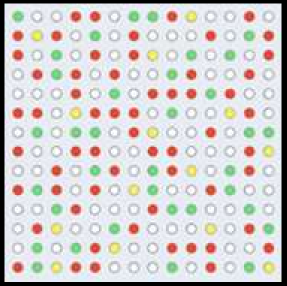
- targets may be **hard to obtain / boring to generate**



Saturn's moon, Titan

https://ai.jpl.nasa.gov/public/papers/hayden_isairas2010_onboard.pdf

- targets may **just not be known**



unsupervised learning

- **unlabeled data**
- learning **without** targets
- **data itself** is used by the algorithm to **guide learning**

- **spotting similarity between various data points**
    - exploit similarity to **cluster similar data points together**
    - automatic classification!

external error function?

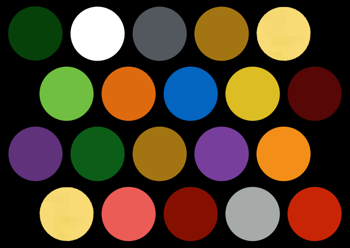
since there is **no target**, there is **no task specific error function**

competitive learning

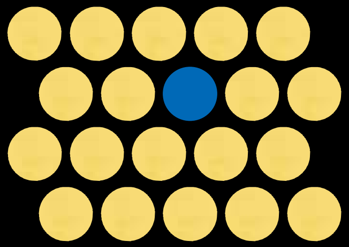
usual practice is to cluster data together via **“competitive learning”**

e.g.

set of neurons



fire the neuron that **best matches (has highest activation w.r.t.)** the **data point/input** ●



let us look at two unsupervised learning algorithms k-means clustering?

self organising maps?

k-means clustering

k-means clustering

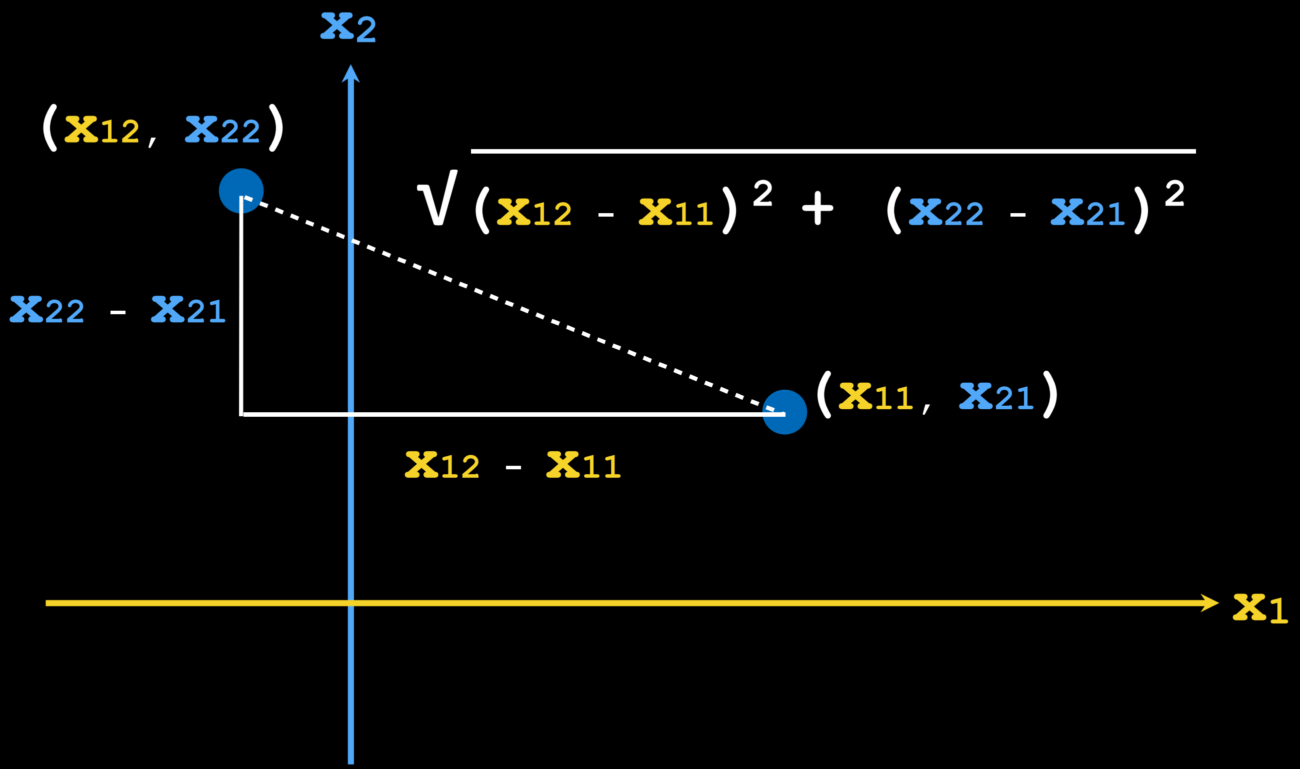
- say **you know the number of clusters** in a data set, but **do not know which data point belongs to which cluster**
- how would you assign a data point to one of the clusters?

flow of k-means

- position **k centers (or centroids)** at random in the **data space**
- **assign each data point** to the **nearest center** according to a **chosen distance measure**
- **move the centers** to the **means of the points** they represent
- iterate

chosen distance measure?

typically euclidean distance



$$\sqrt{(x_{12} - x_{11})^2 + (x_{22} - x_{21})^2}$$

k?

- **k points** are used to represent the clustering result, **each such point** being the **mean of a cluster**
- **k must be specified**

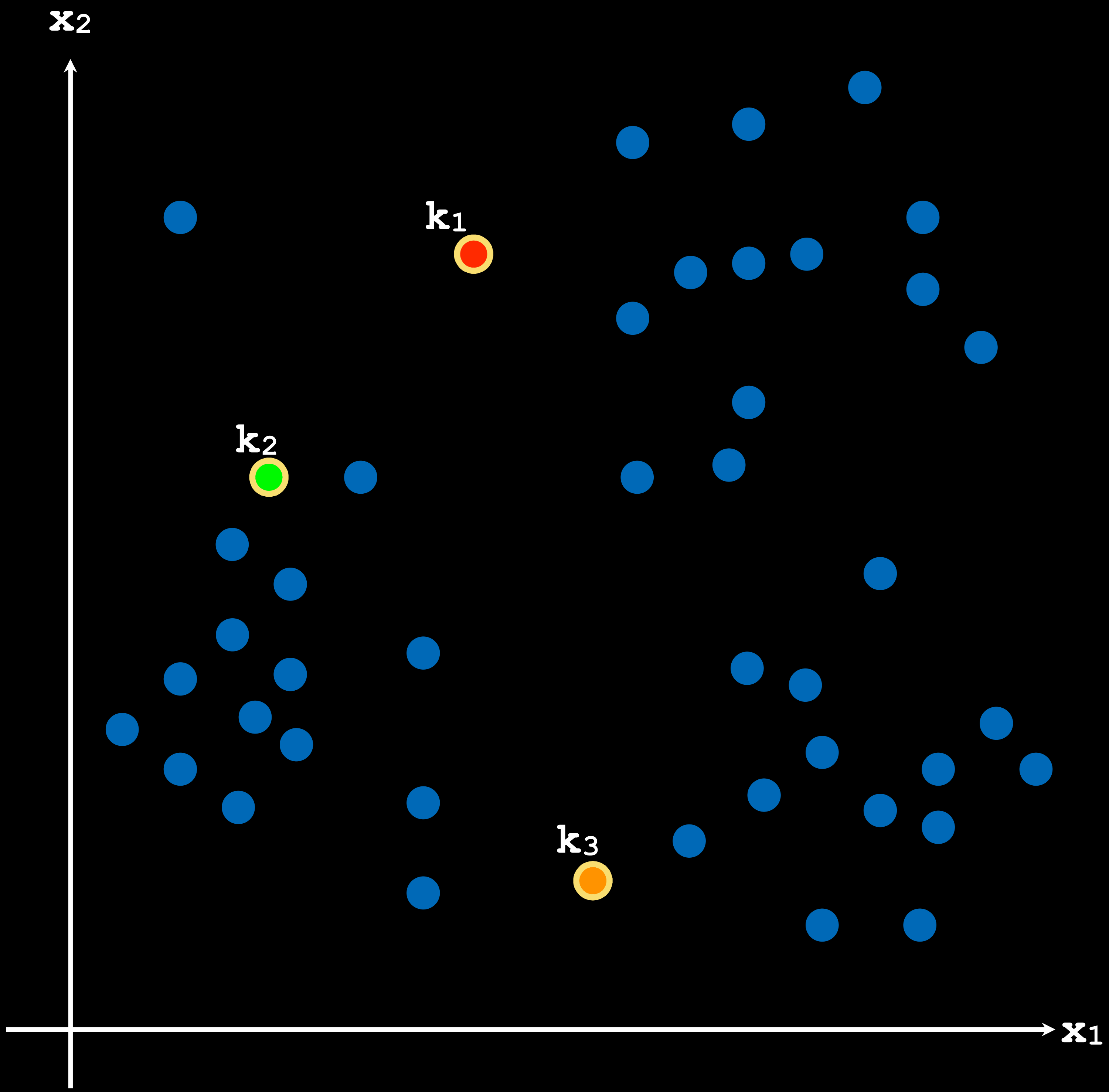
the algorithm

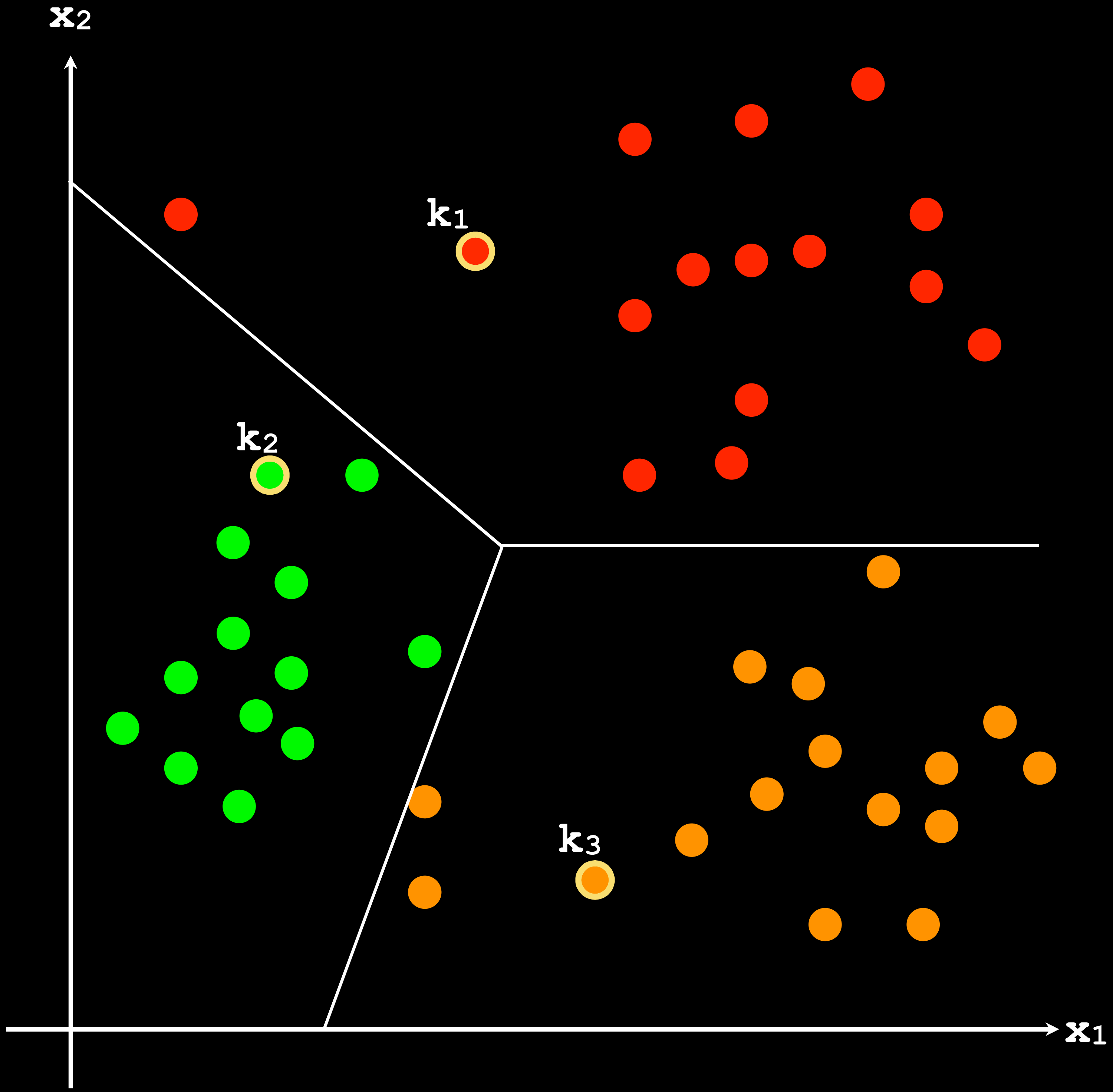
(1) **pick** a number, **k**, of **cluster centers** (at random, do not have to be data points)

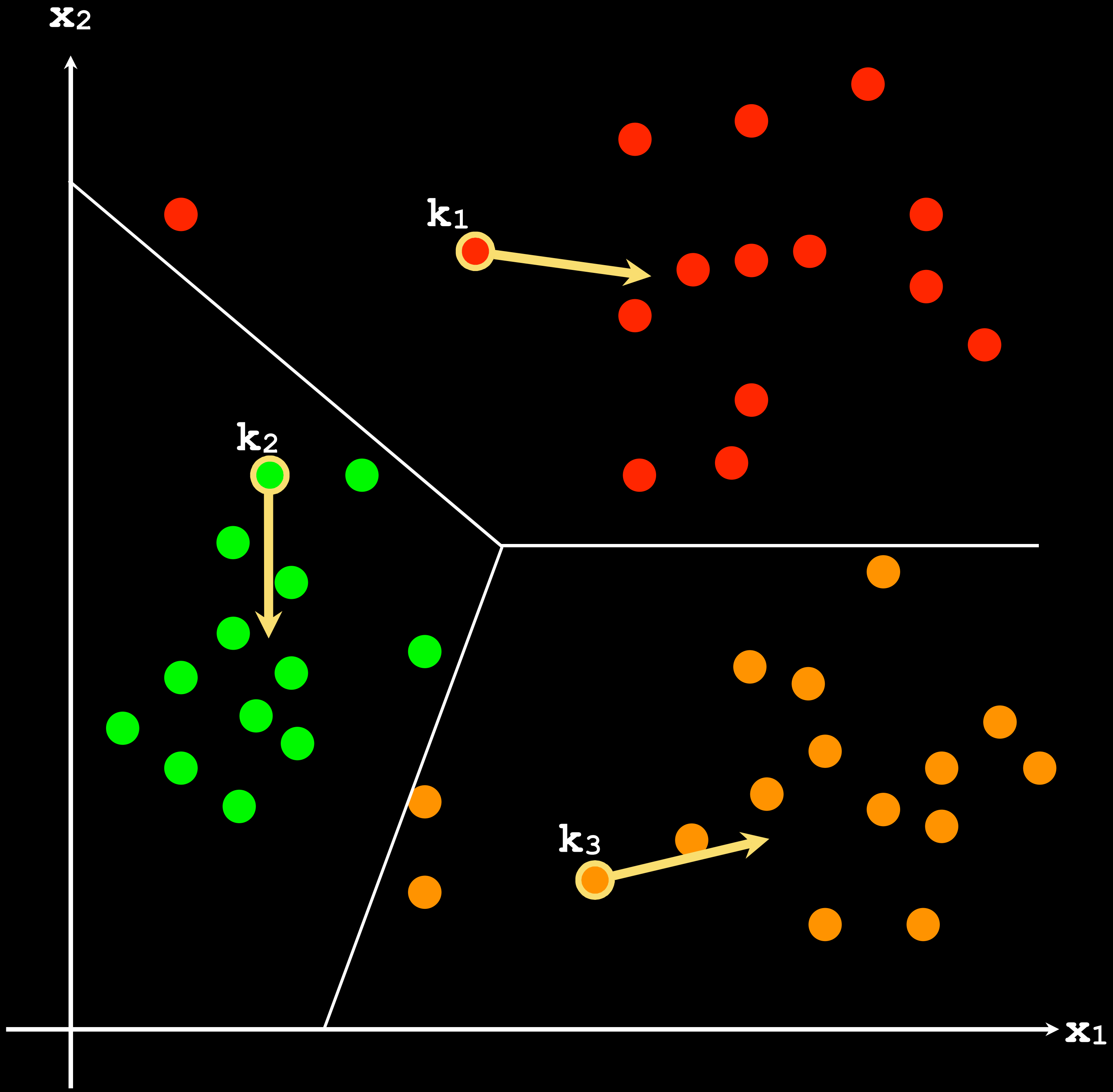
(2) **assign** every **data point** to its **nearest cluster center** (e.g. using euclidean distance)

(3) **move** each **cluster center** to the **mean** of data points assigned to it

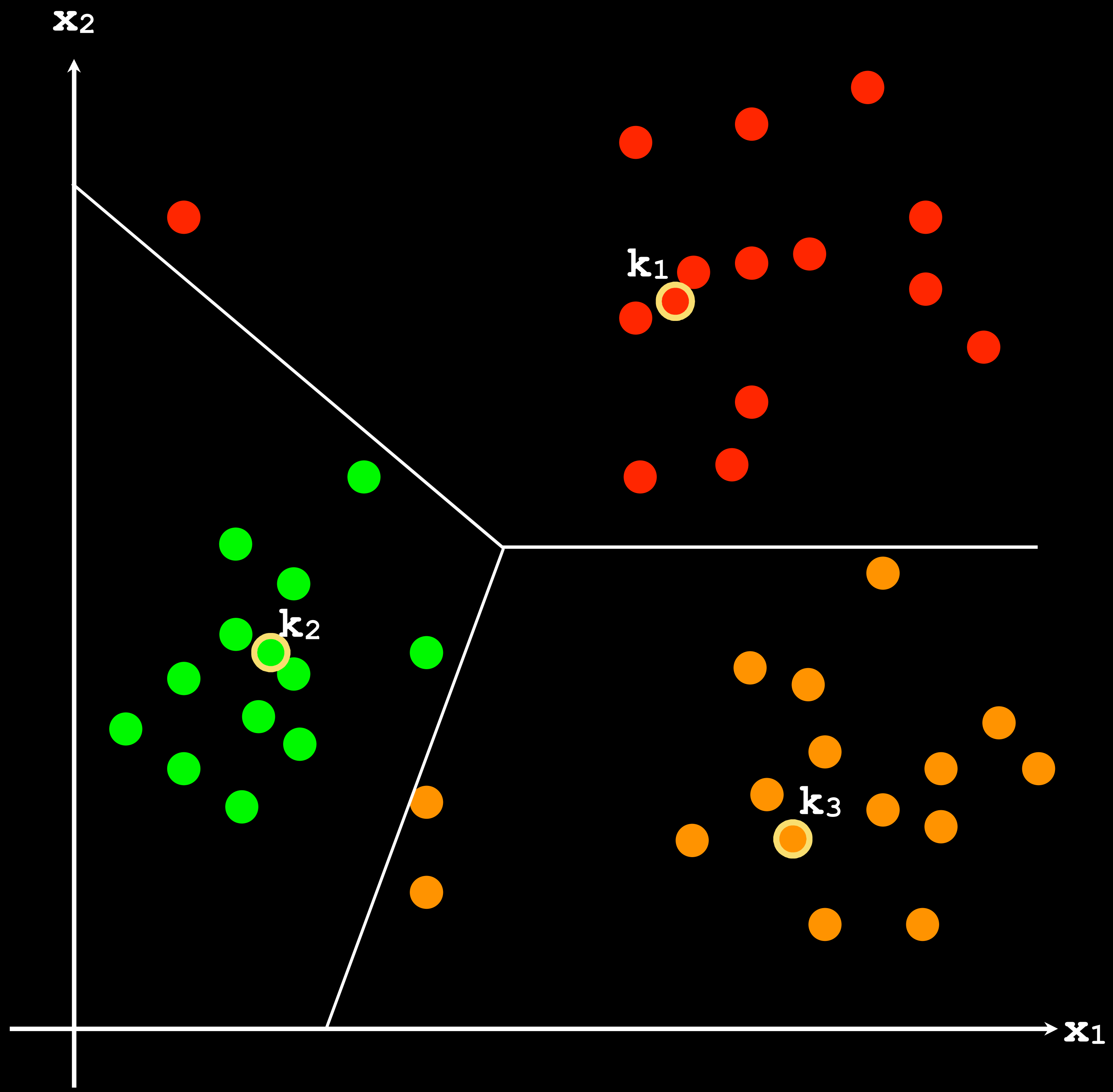
(4) **repeat** steps (2) and (3) until **convergence** (e.g. change in cluster assignments less than a threshold)

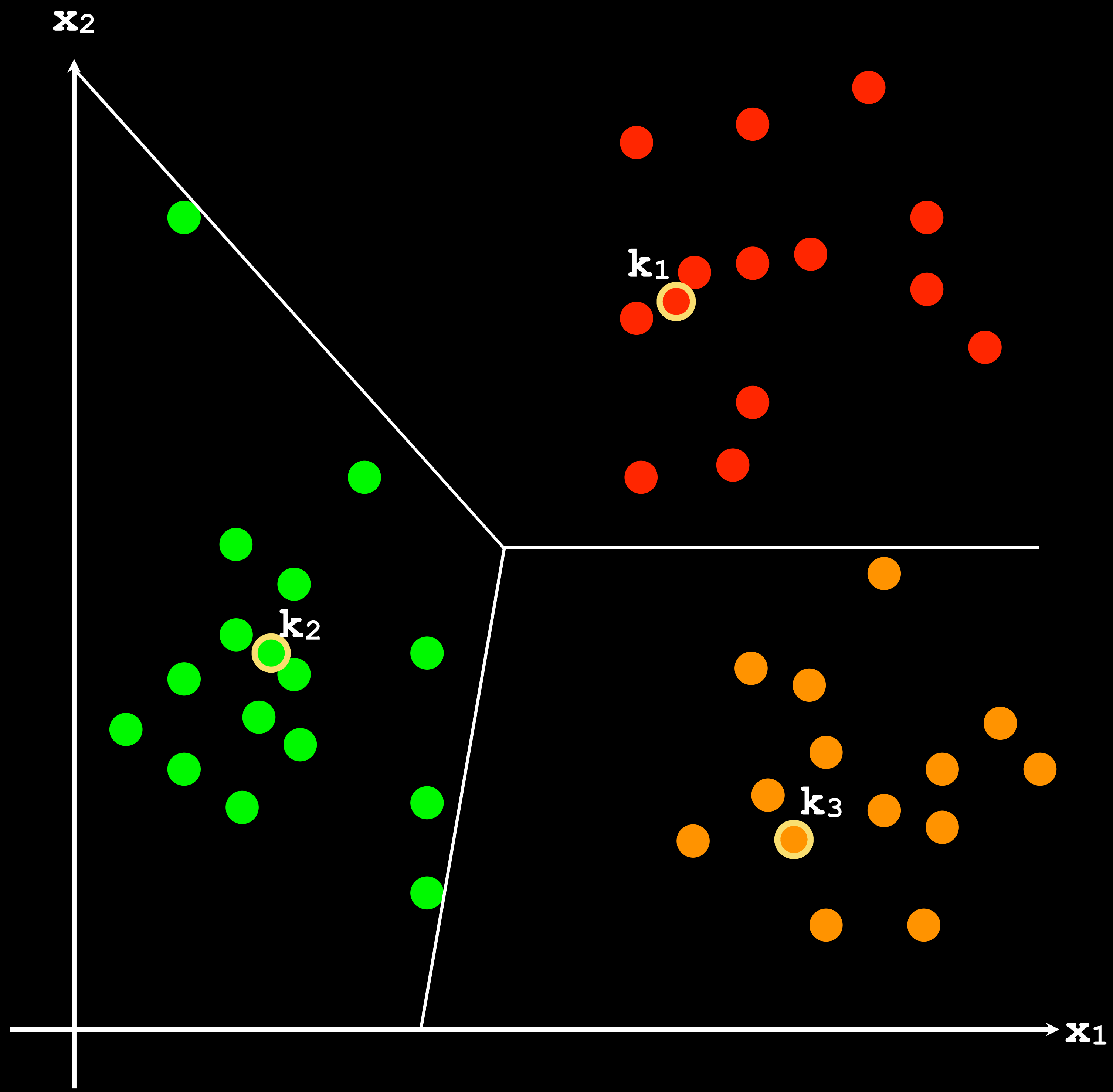

$\mathbf{x}_2$
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{x}_1$

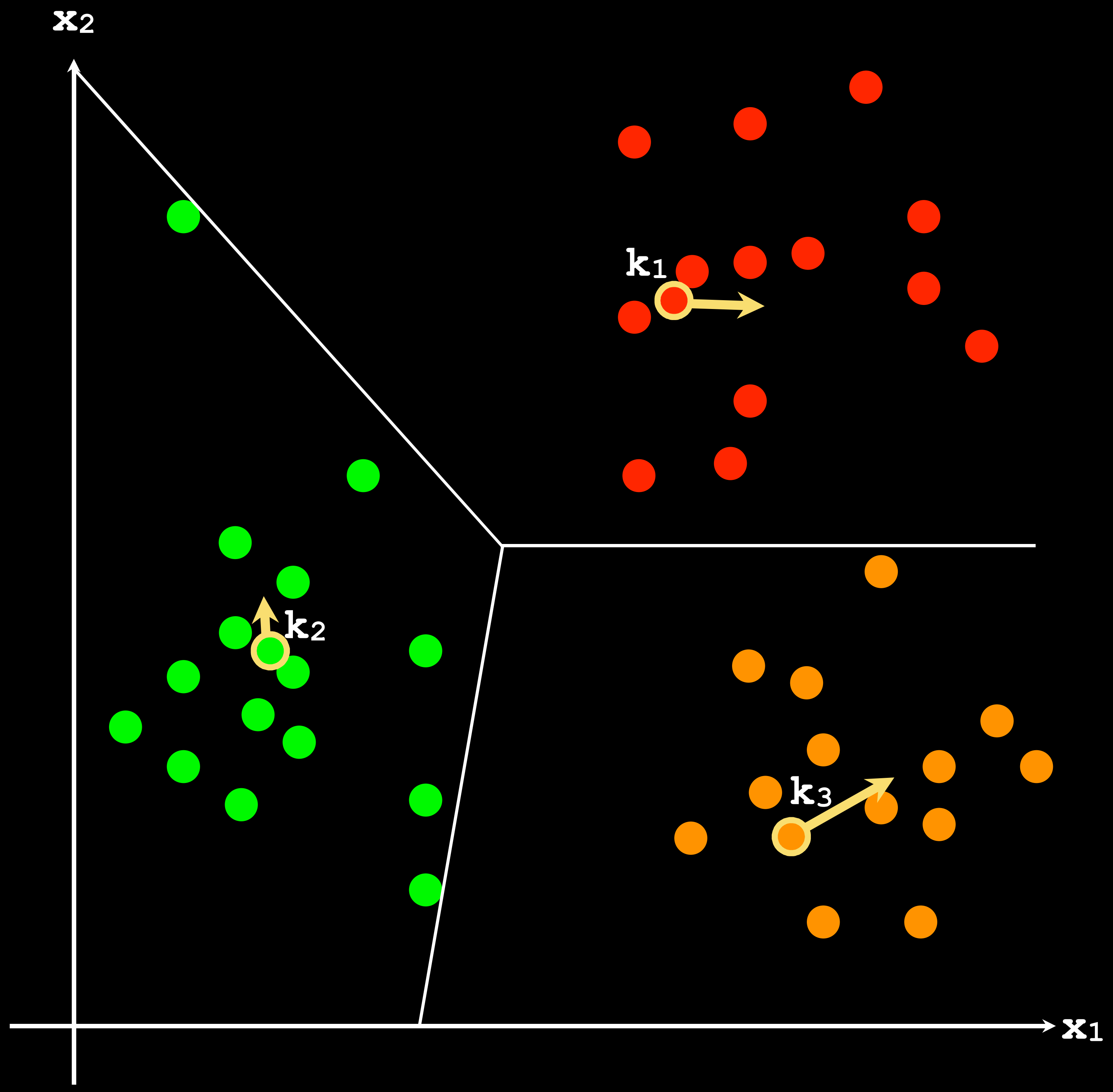

x_2
k_1
k_2
k_3
x_1

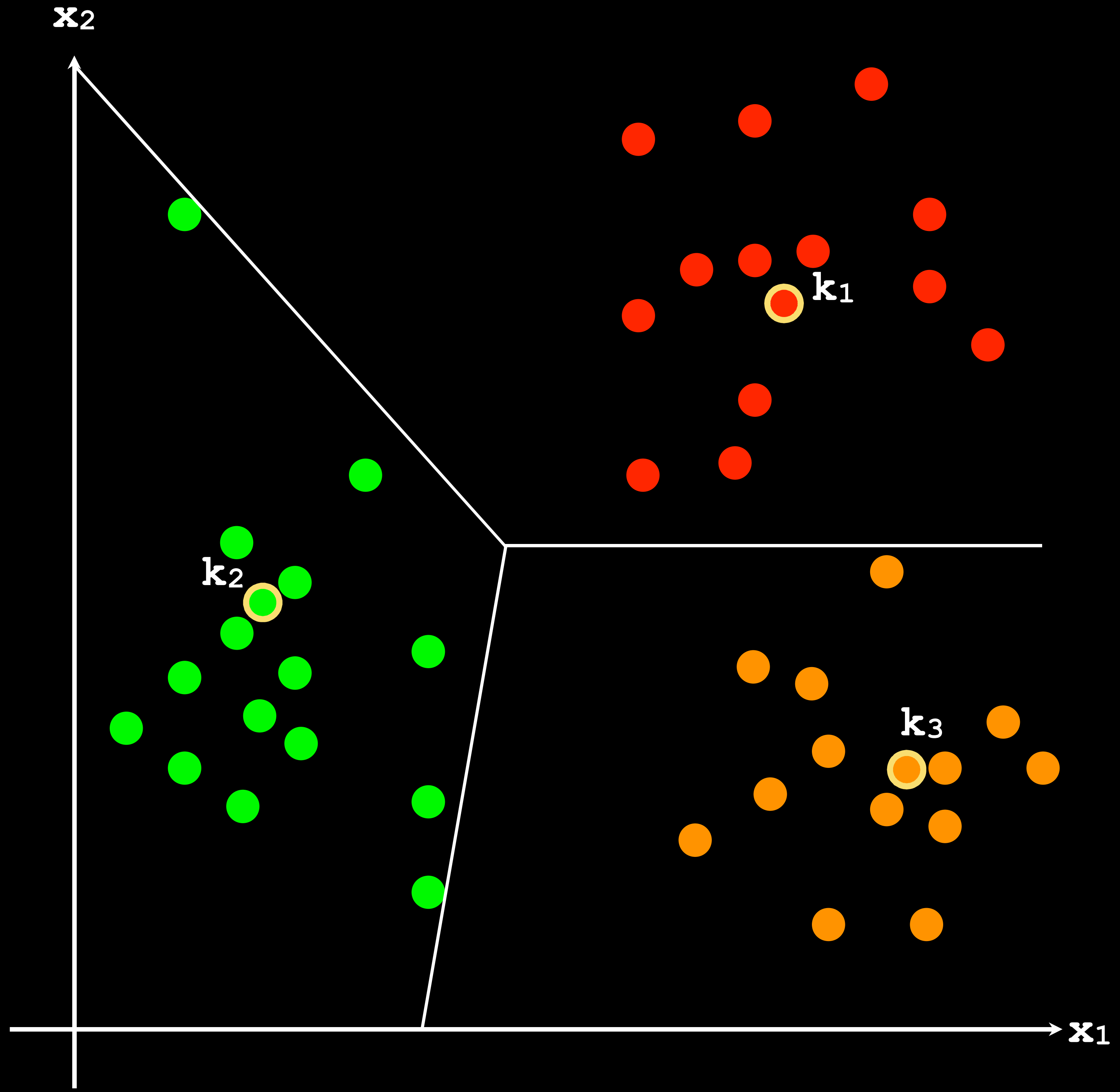


x_2
k_1
k_2
k_3
x_1


x_2
k_1
k_2
k_3
x_1

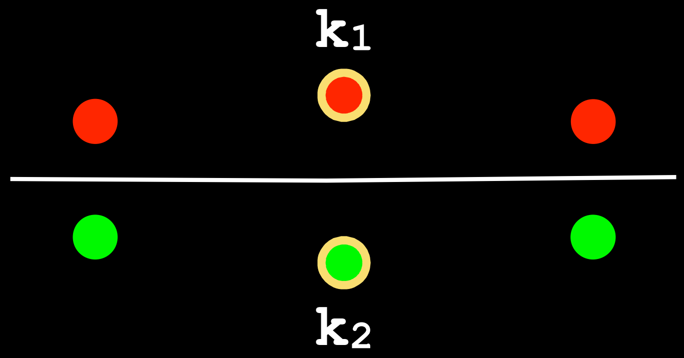

x₂
k₁
k₂
k₃
x₁


$\mathbf{x}_2$
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{x}_1$


x2
k1
k2
k3
x1

some thoughts...

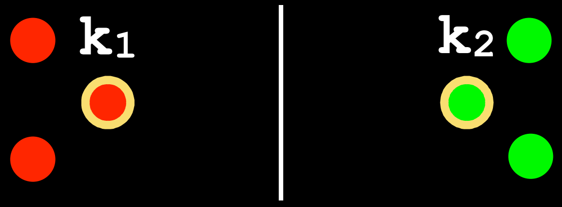
- results vary depending on **initial choice of cluster centers**
- can be **trapped in local minima**
  - restart with different random centers



- does not handle **outliers** well

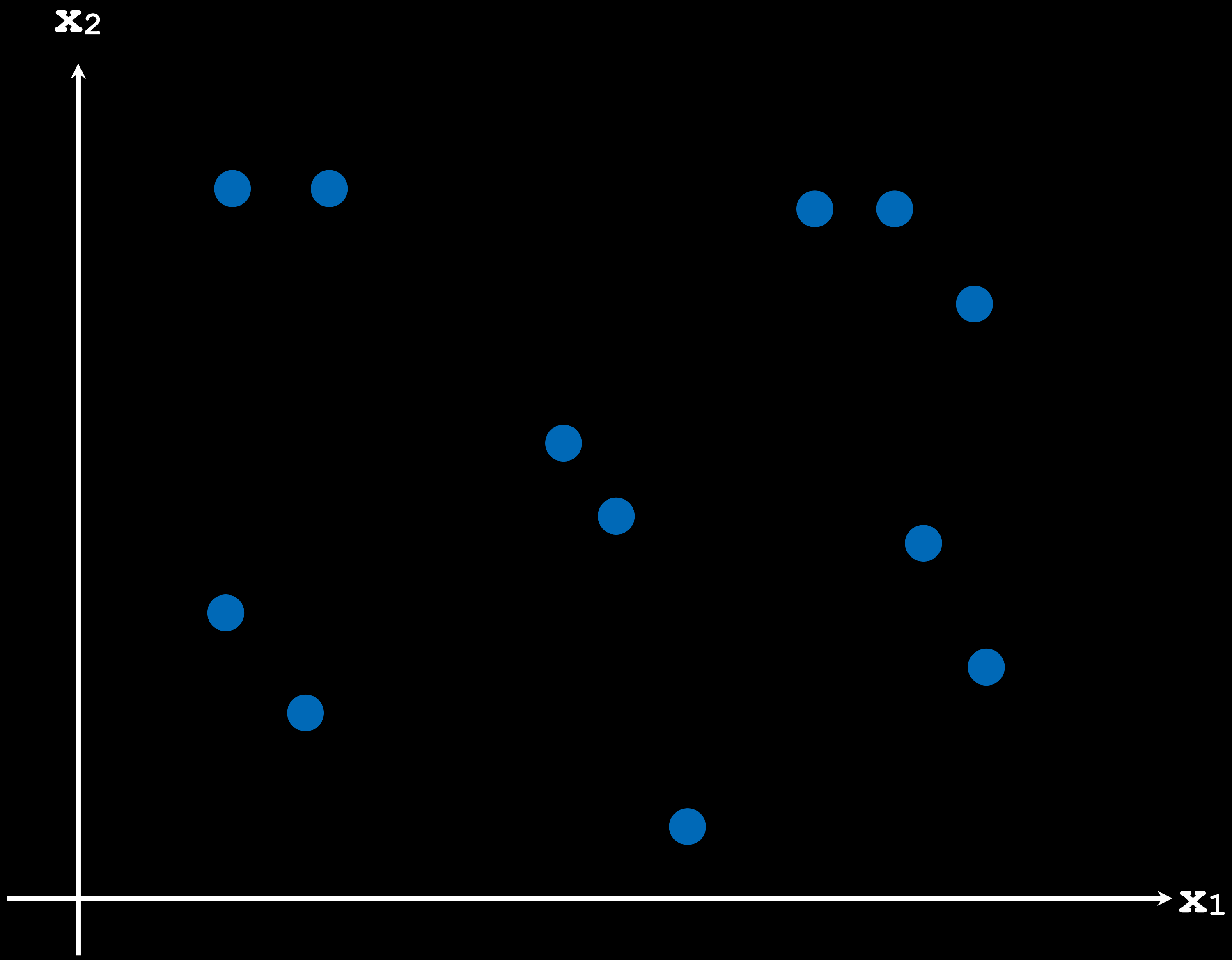
some thoughts...

- results vary depending on **initial choice of cluster centers**
- can be **trapped in local minima**
  - restart with different random centers

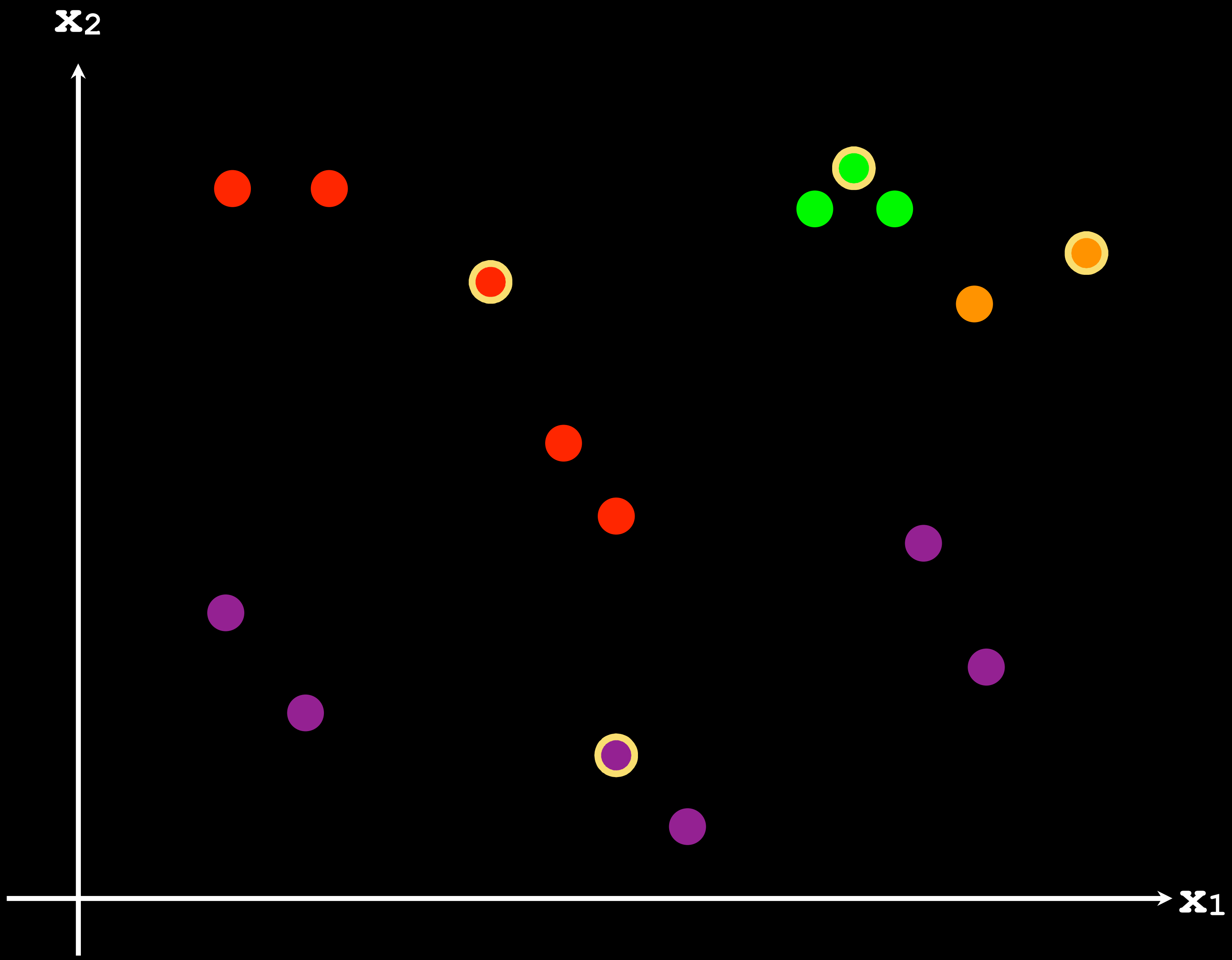


- does not handle **outliers** well

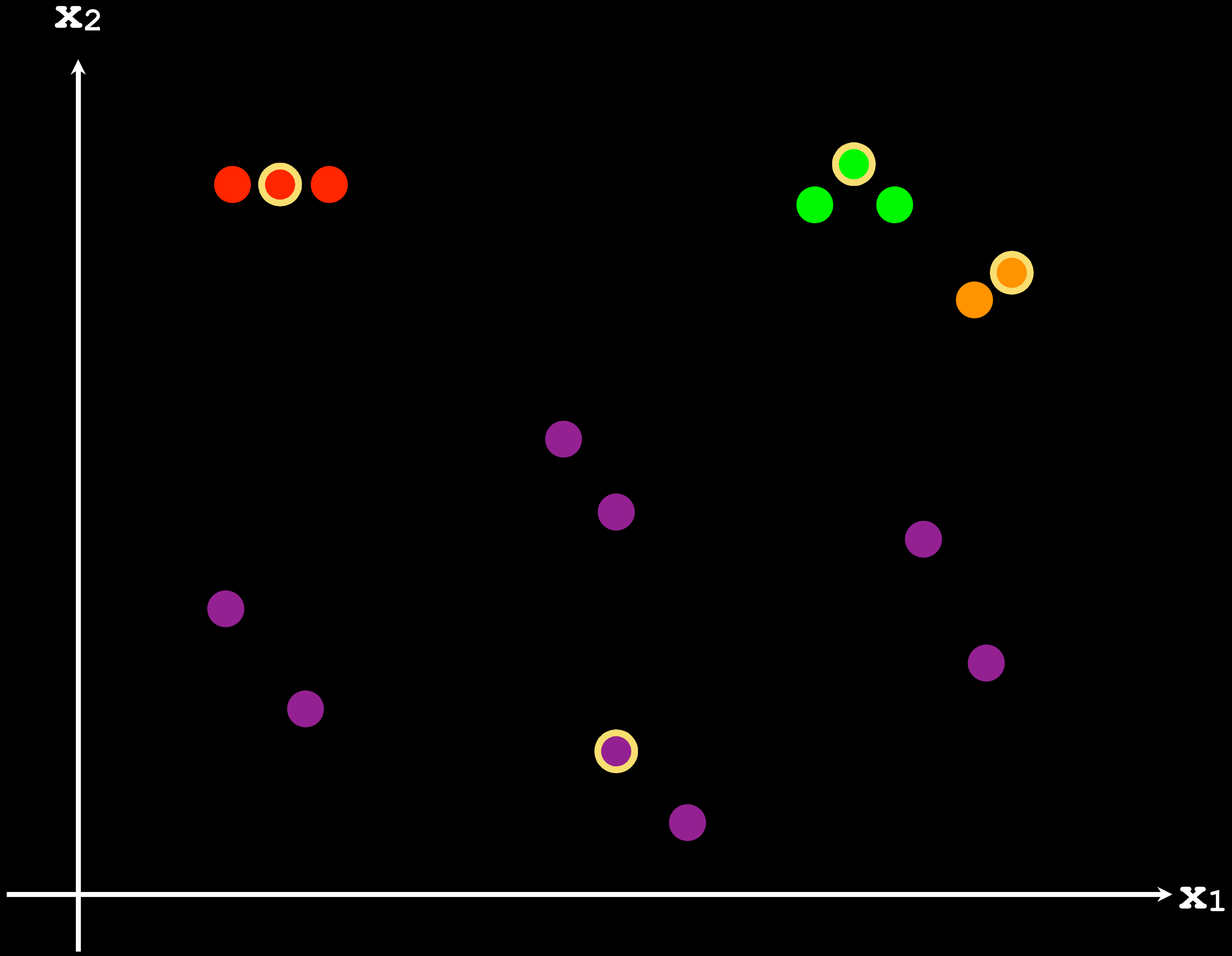
let’s look at the dependence on initial choice…



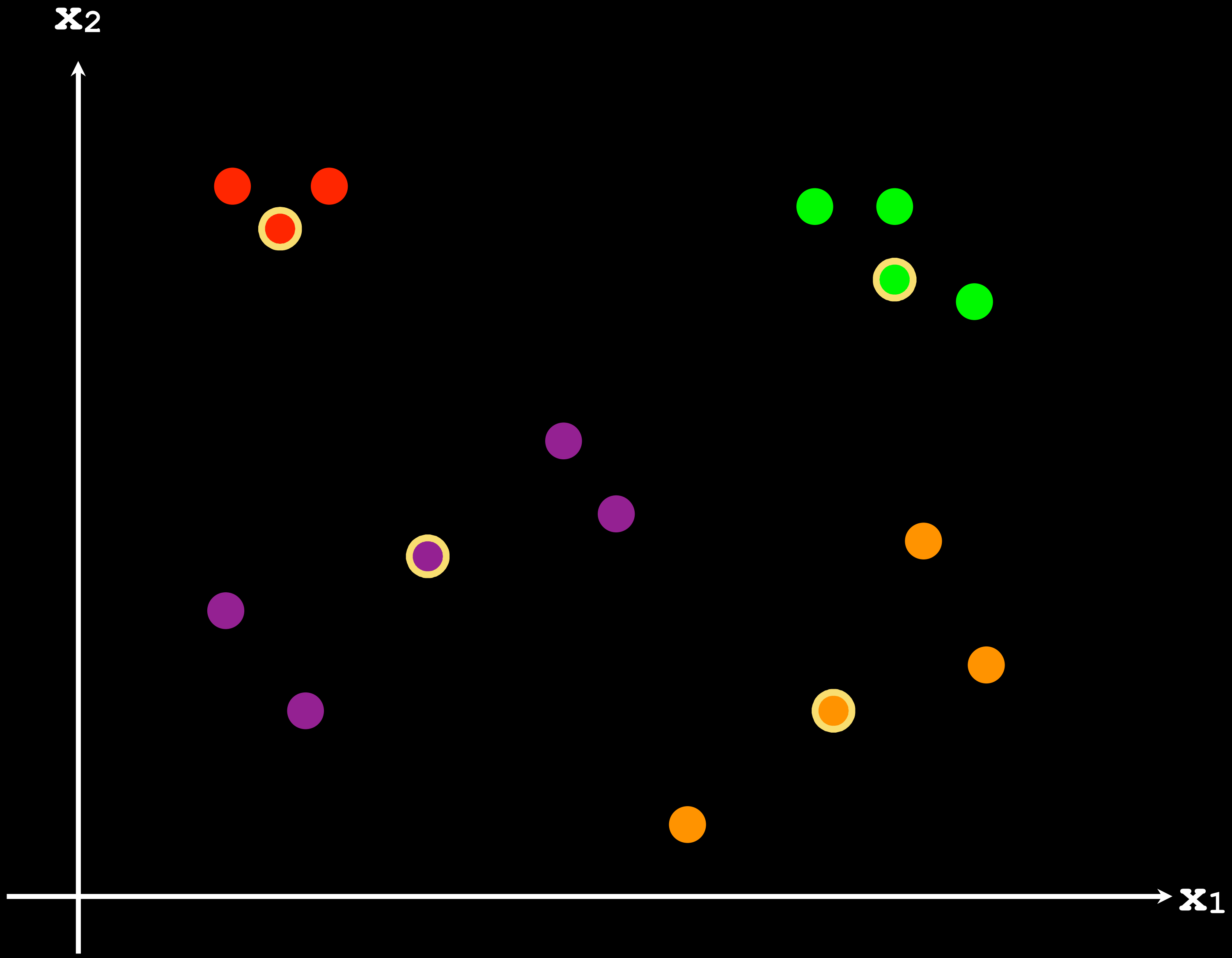
a solution...



another solution...



yet another solution...

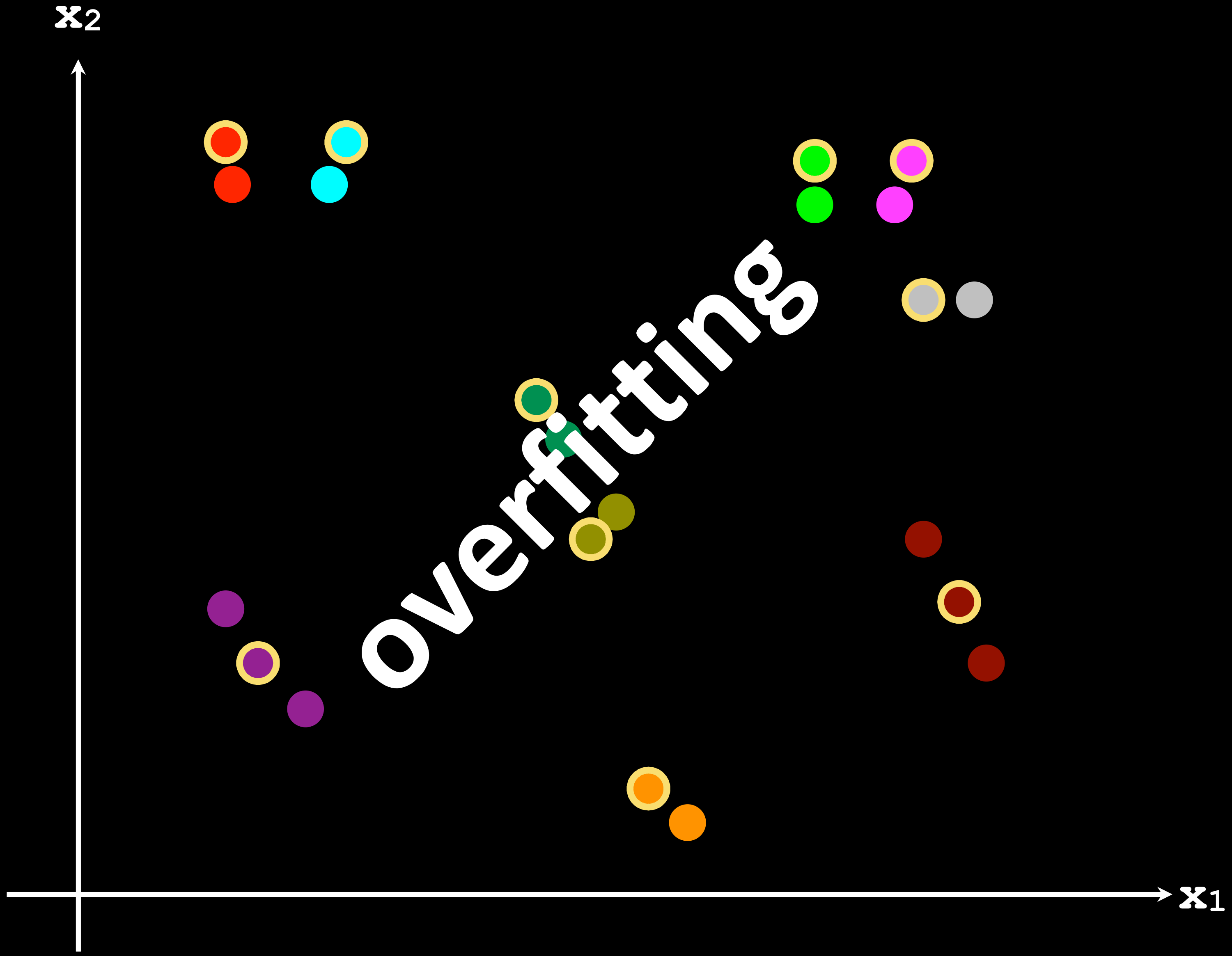


all these solutions are

local minima!

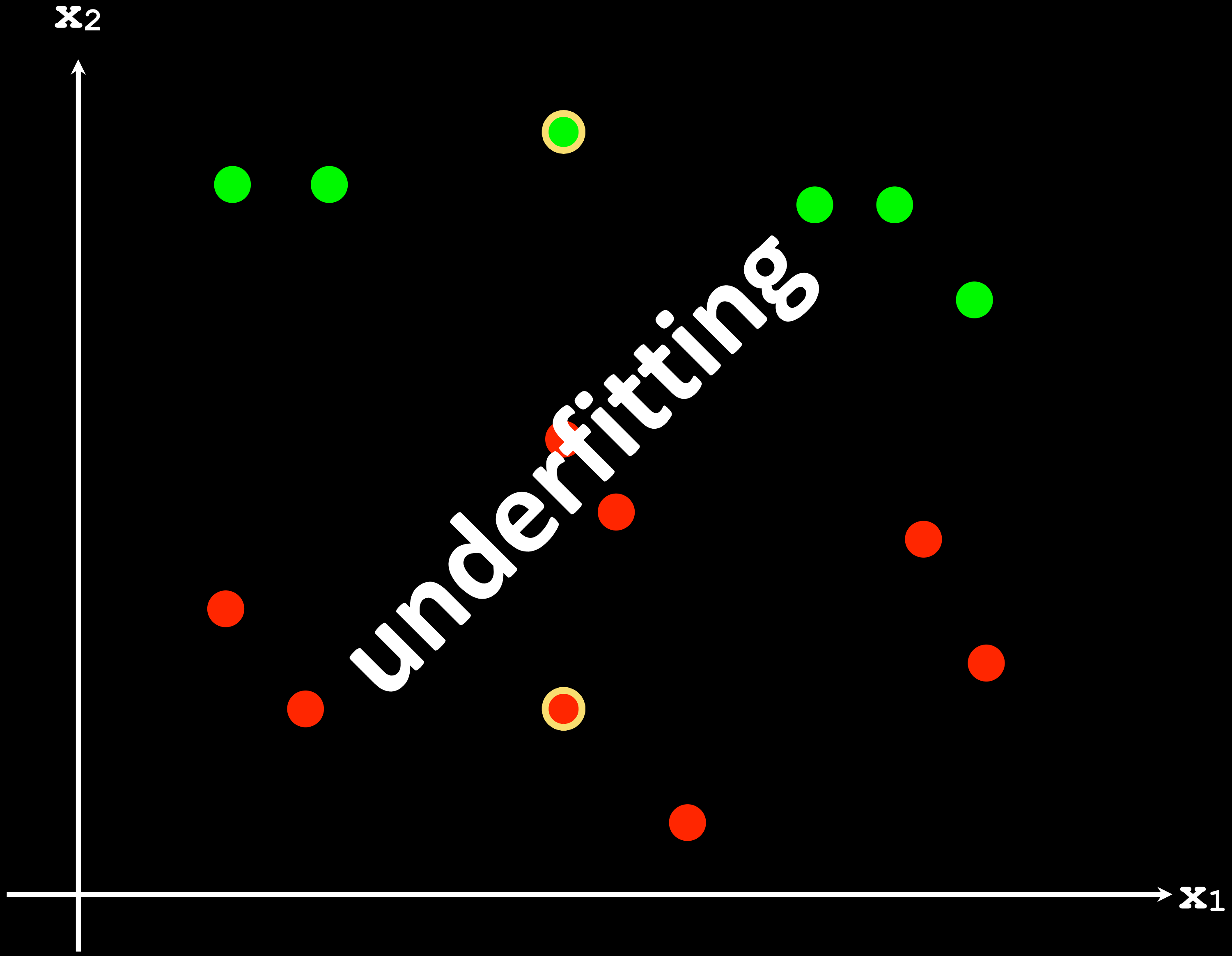
choice of k

not knowing k leads to further problems!



choice of k

not knowing k leads to further problems!



what are we minimising?

- there is no externally given error function
- the **within cluster sum of squared** error is what k-means tries to minimise
- so, with k clusters $K_1, K_2, \ldots, K_k$, centers $k_1, k_2, \ldots, k_k$, and data points x_js, we effectively minimise:

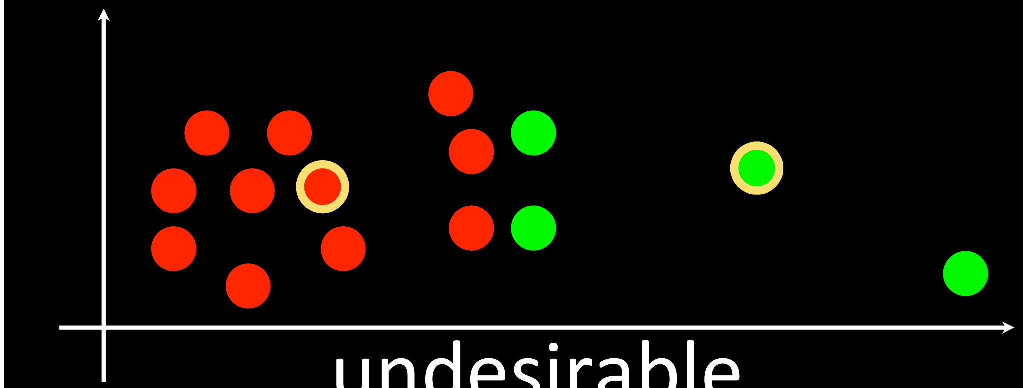
$$\sum_{i=1}^{k} \sum_{x_j \in K_i} \left\| x_j - k_i \right\|^2$$

possible remedies...

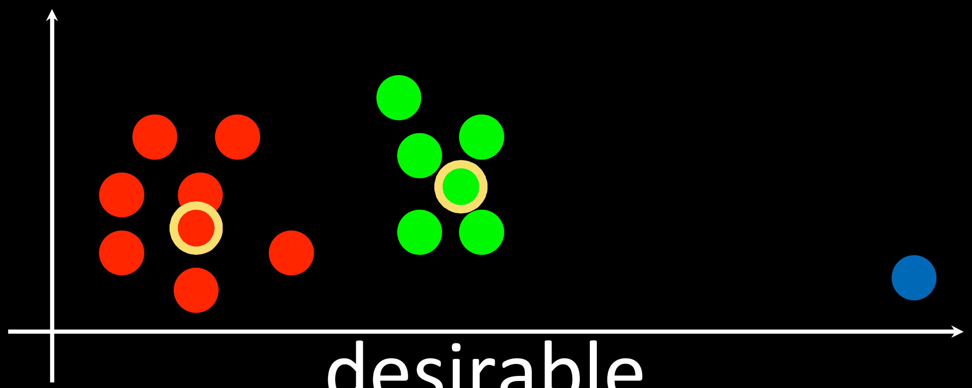
- run algorithm many times with **different values of k**
  - pick k that leads to lowest error without overfitting
- run algorithm from **many starting points**
  - to avoid local minima

noise?

- mean susceptible to outliers (very noisy data)
- one idea is to **replace mean by median**
- 1,2,1,2,100?
  - mean: 21.2 (affected)
  - median: 2 (not affected)



undesirable



desirable

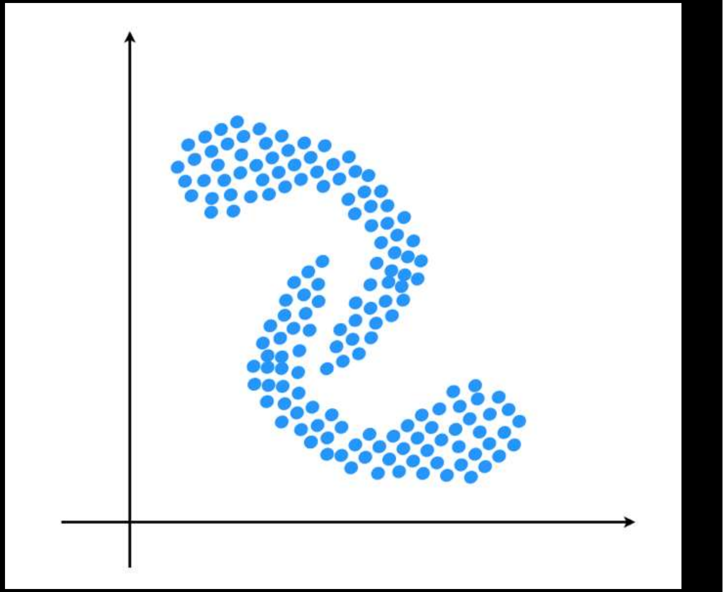
strengths of k-means?

- **simple**: easy to understand and implement
- **efficient** with time complexity O(tkn)
  - n = #data points, k = #clusters, t = #iterations
- typically, k and t are small, so considered a **linear algorithm**

weaknesses?

- unable to handle **noisy data/outliers**

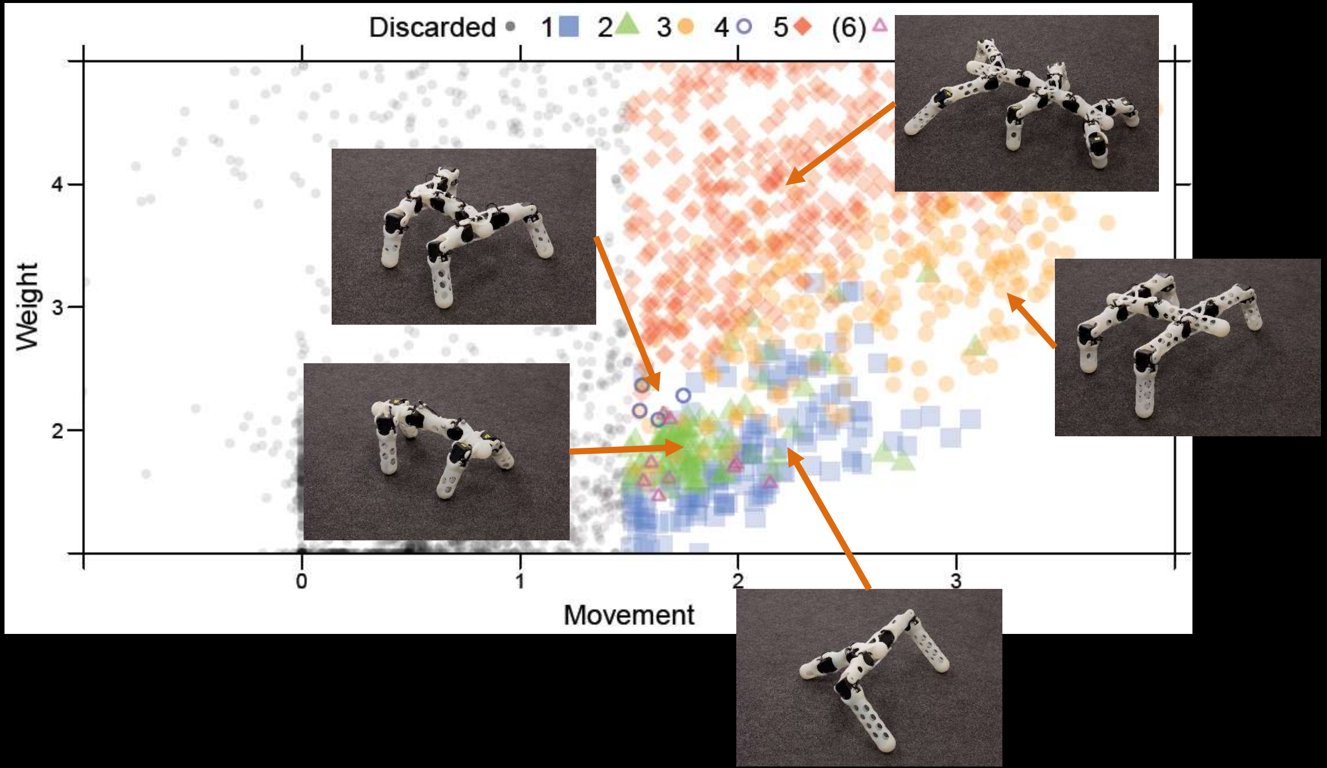
- unsuitable for discovering clusters with **non-convex shapes**



- k has to be **specified in advance**

clustering example: evolutionary robotics

- 949 robot solutions from simulation
- identify a small number of representative shapes for producution



self-organising maps

self-organising maps

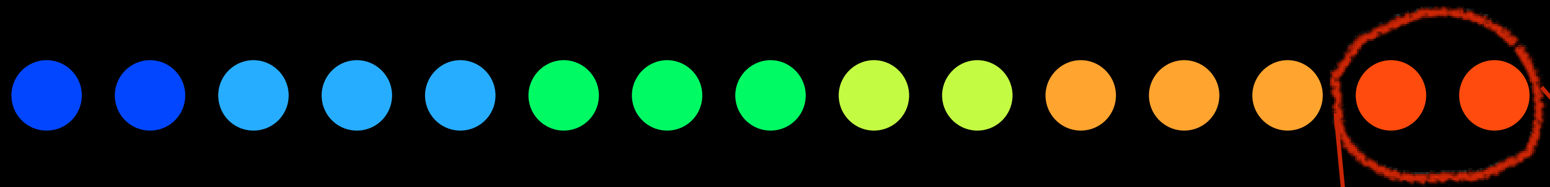
- high dimensional **data hard to understand** as is
- **data visualisation and clustering** technique that **reduces dimensions of data**
- **reduce dimensions** by **projecting and displaying the similarities** between data points on a **1 or 2 dimensional map**

a neural network with topological meaning

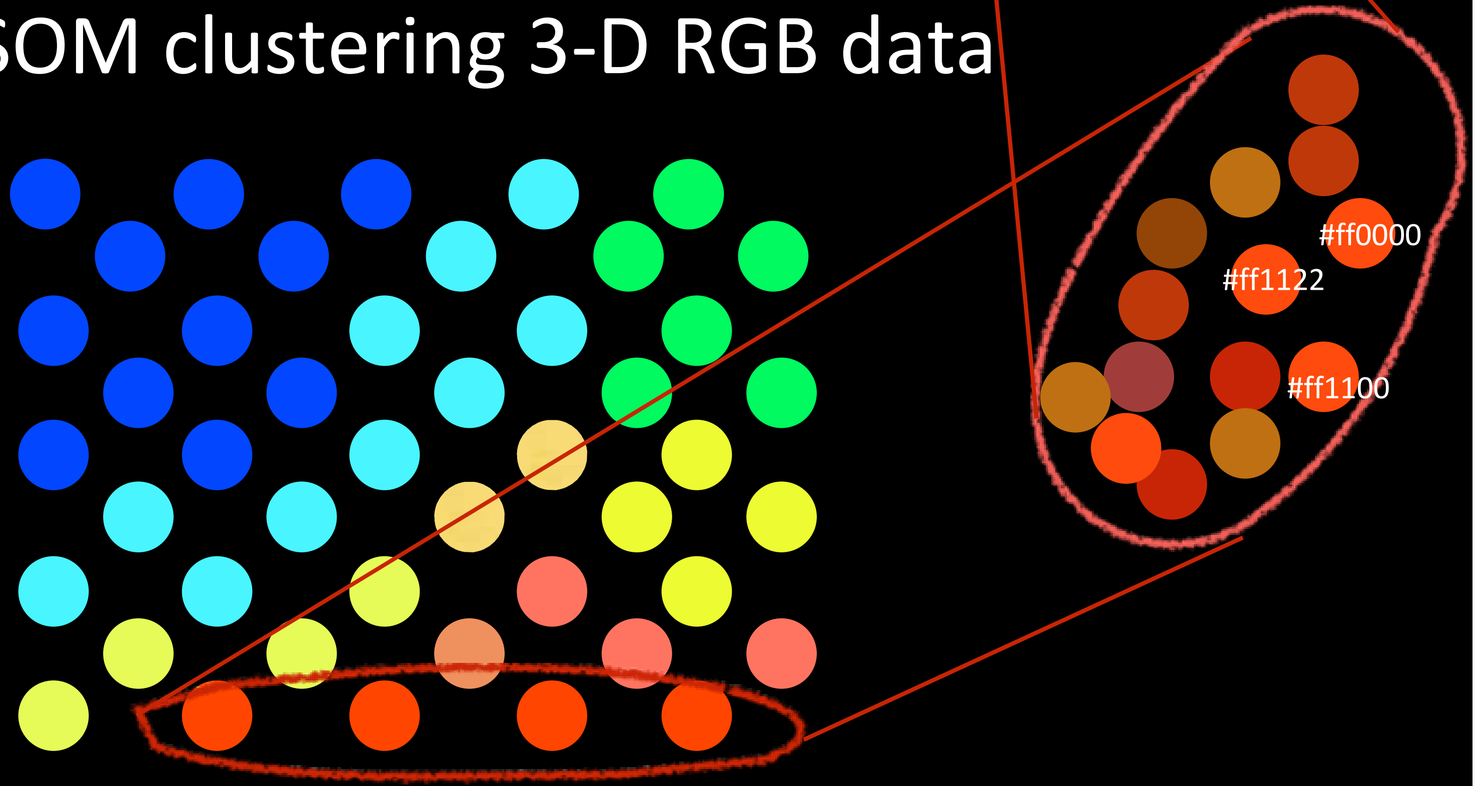
- a SOM is an artificial neural network trained in an unsupervised manner
- the network is able to cluster data in a way that **topological relationships between data points are preserved**
  - i.e. **neurons close together** **represent** **data points that are close together**

e.g.

1-D SOM clustering 3-D RGB data



2-D SOM clustering 3-D RGB data



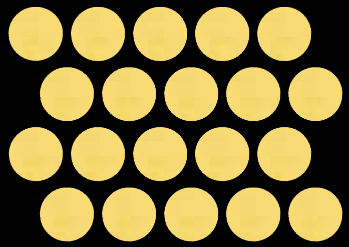
biological motivation...

- motivated by how visual, auditory, and other **sensory information** is **handled in separate parts of the cerebral cortex** in the human brain
- sounds that are **similar** excite neurons that are **near to each other**
- sounds that are **very different** excite neurons that are a **long way off**
- **input feature mapping!**

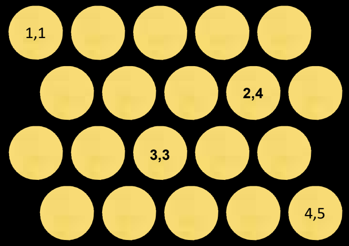
from motivation to inspiration…

- so the idea is that learning should **selectively tune** neurons **close** to each other **to respond to/represent a cluster** of data points
- first described as an ANN by **Prof. Teuvo Kohonen**

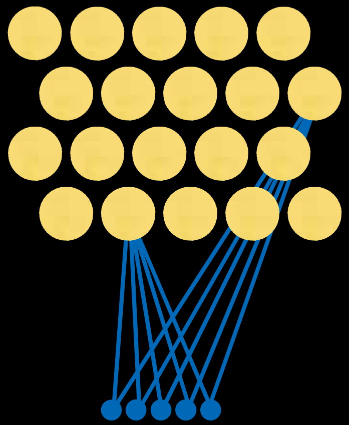
SOM consists of components called nodes/neurons



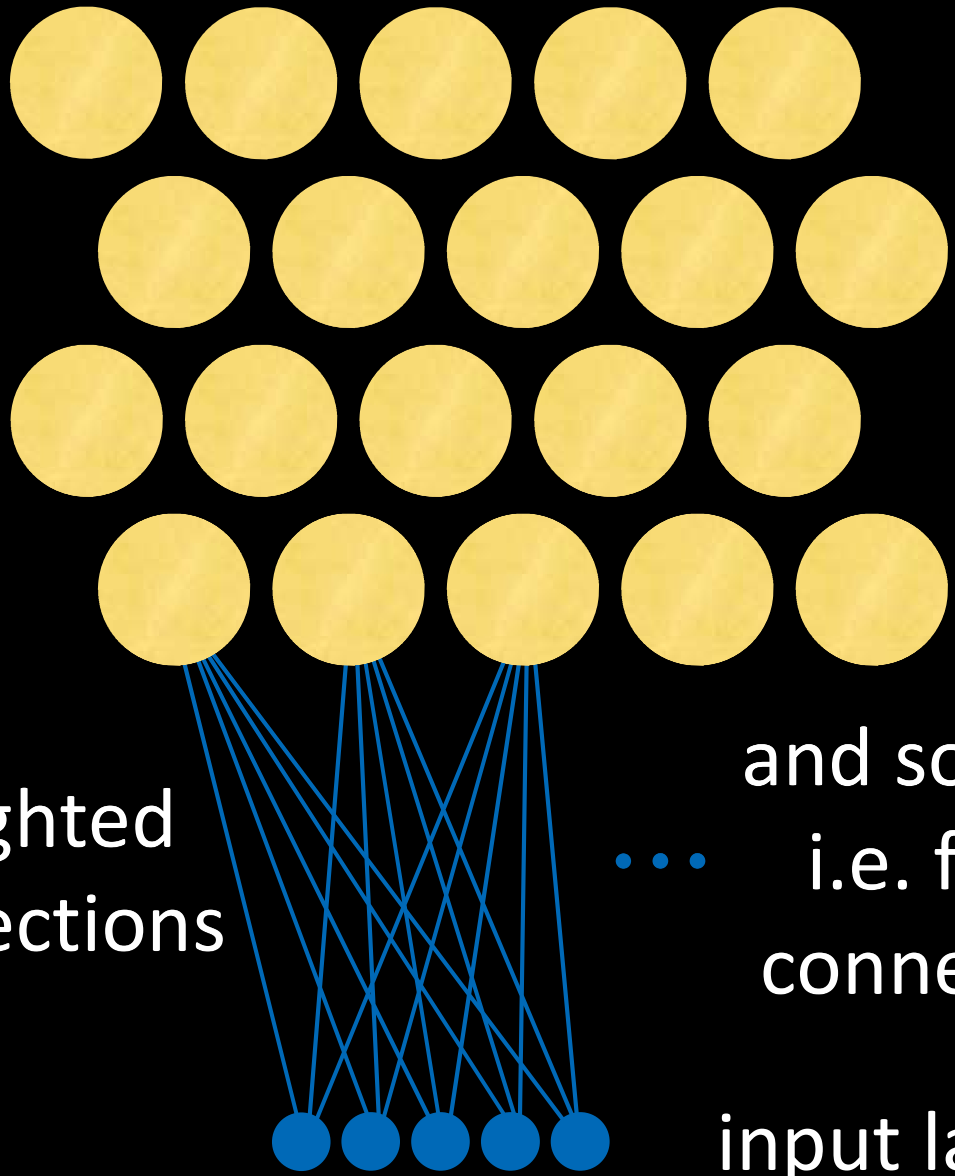
each node has a position associated with it on the map



and a weight vector of dimension given by the data points (input vectors)



e.g. say, 5D input vector

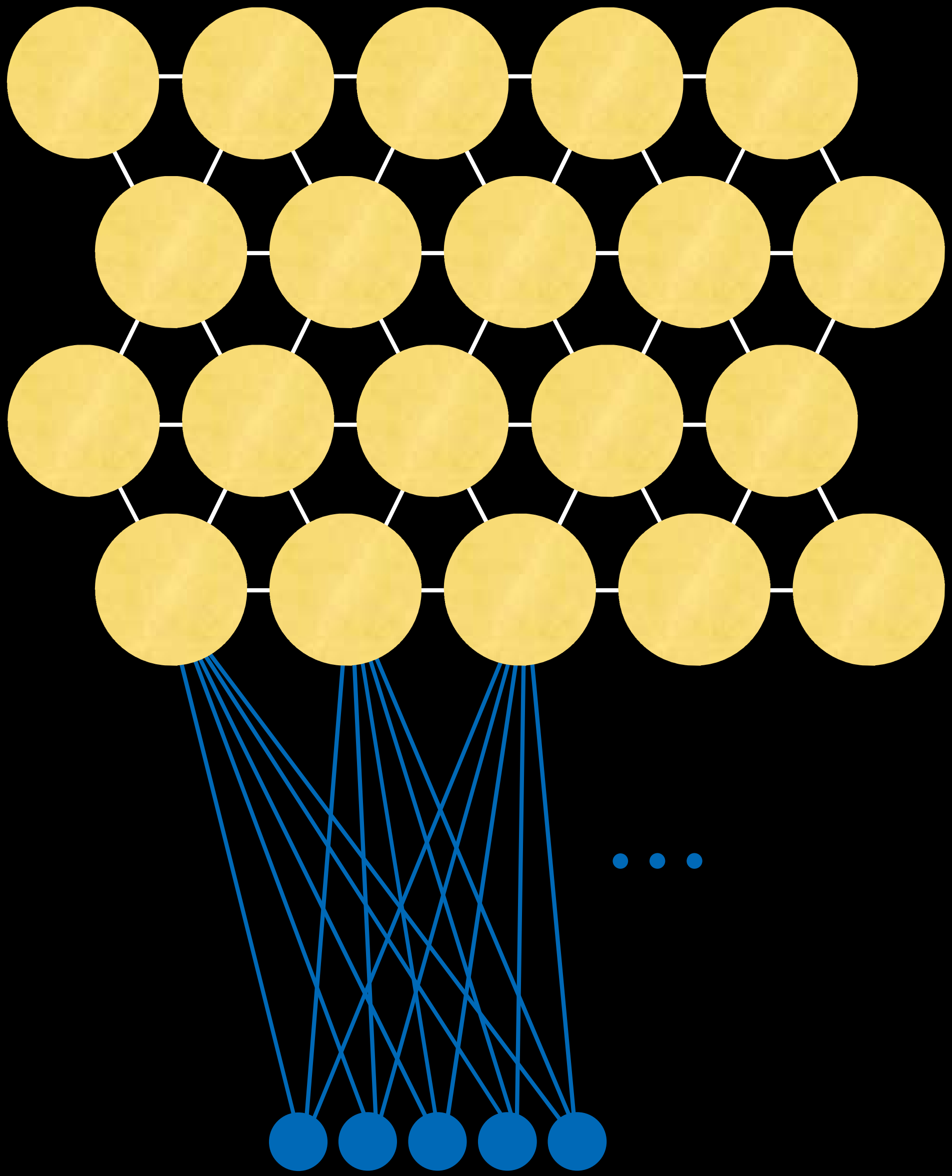


feature/output/
map layer

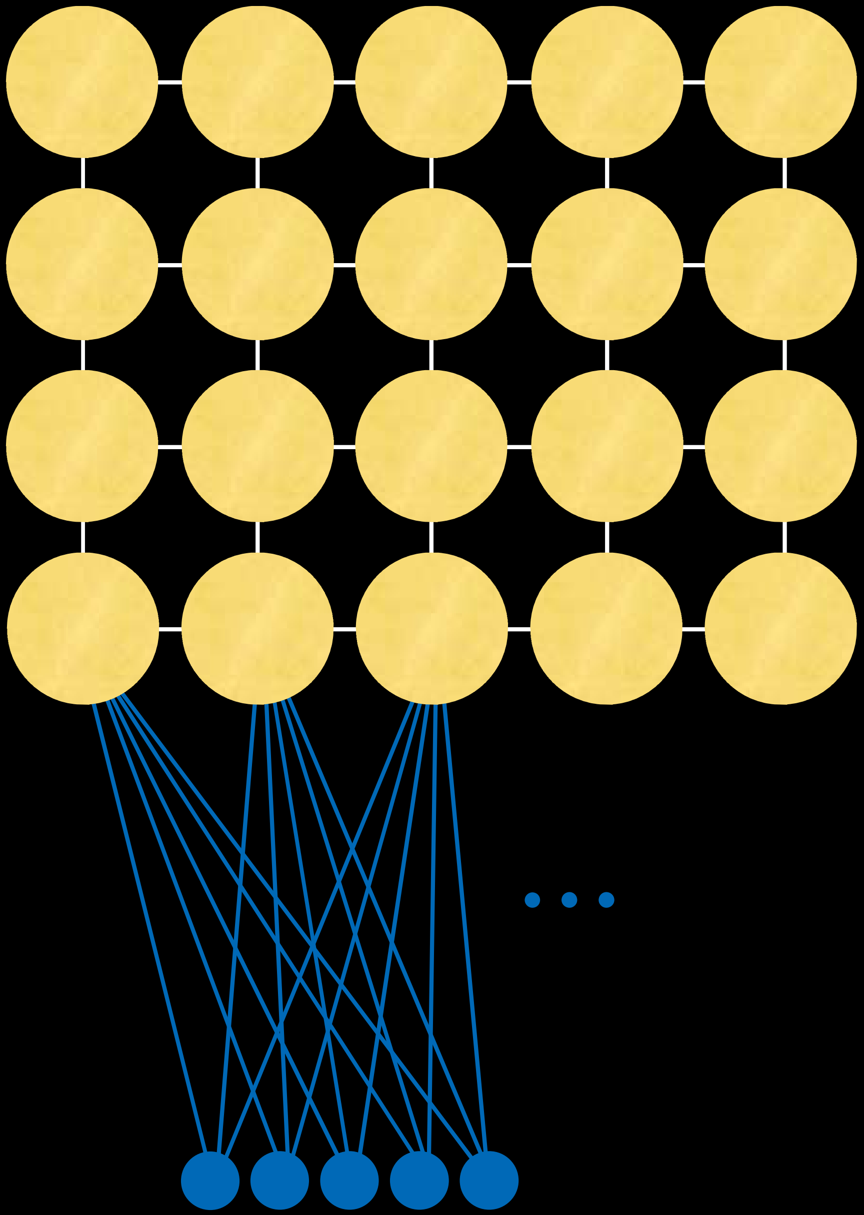
weighted
connections

and so on...
i.e. fully
connected

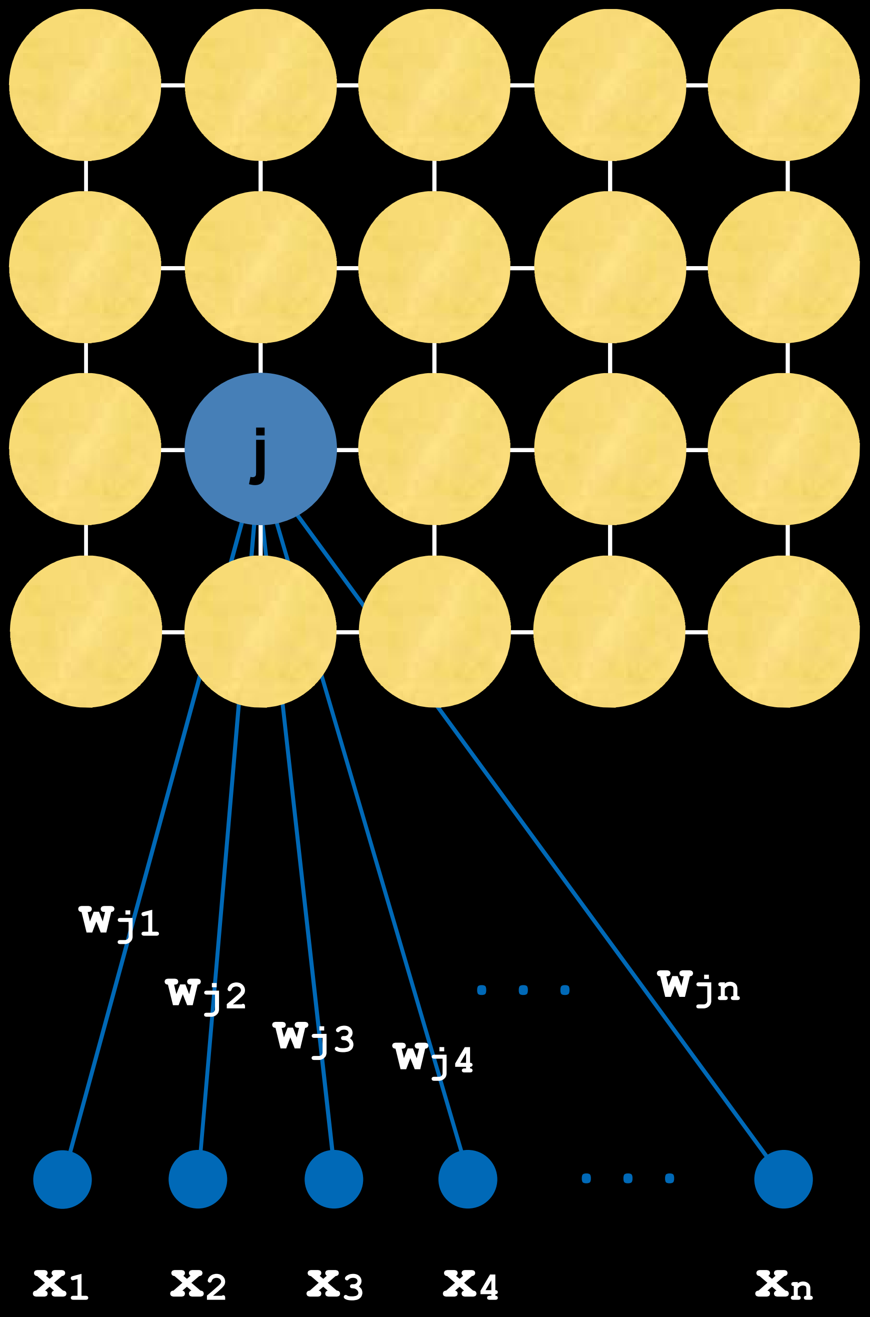
input layer



neurons are interconnected within a defined neighbourhood (hexagonal here) i.e. neighbourhood relation defined on output layer



typically, rectangular or hexagonal lattice neighbourhood/topology for 2D SOMs



lattice responds
to input

one neuron wins,
i.e. has the highest
response
(known as the **best
matching unit**)

matching?

matching?

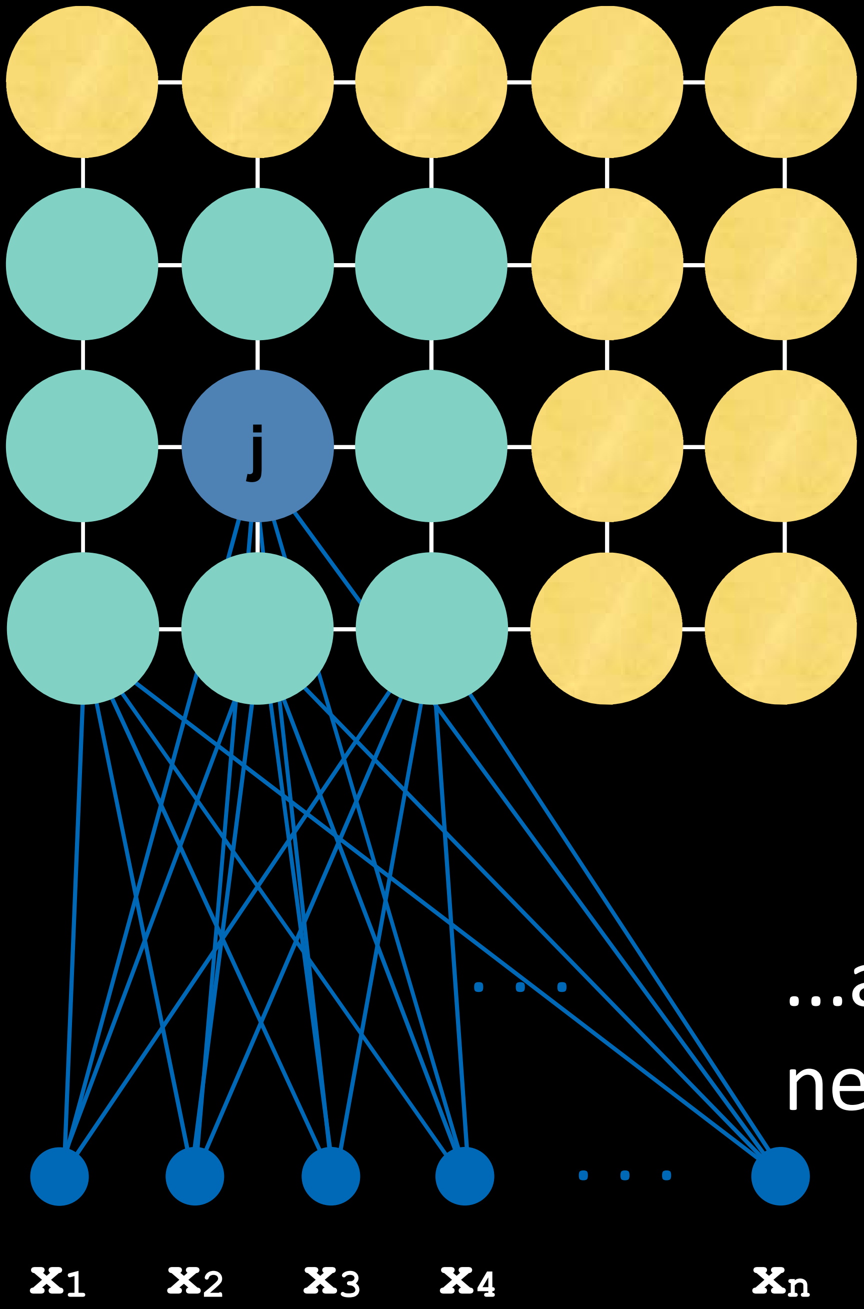
- input and weight vectors can be matched in numerous ways
- typically:

$$\|\mathbf{x} - \mathbf{w}\|$$ **euclidean**

$$\sum_{i=1}^{n} |x_i - w_{ji}|$$ **manhattan**

$$\mathbf{x} \cdot \mathbf{w}$$ **dot product**

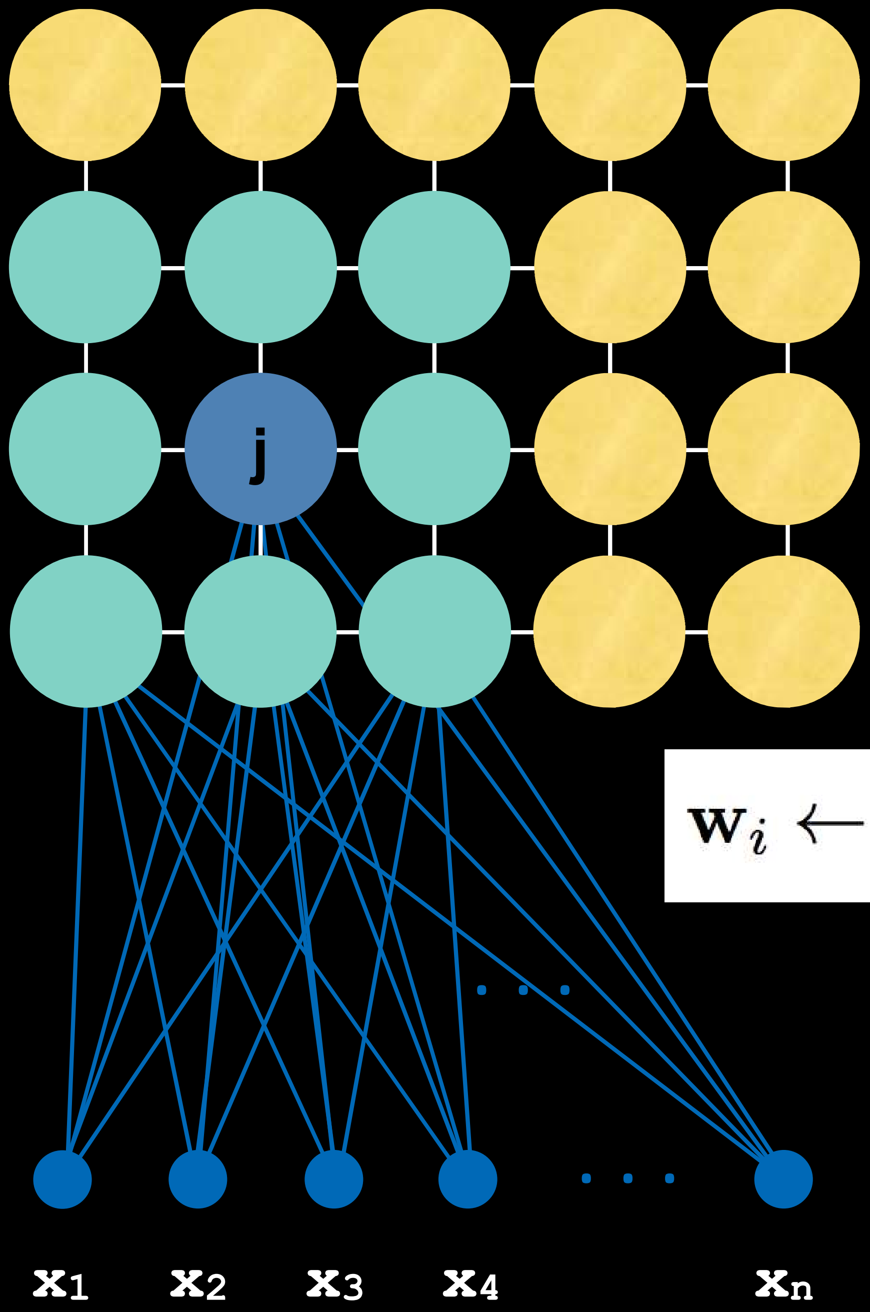
learning...



adapting weights of winner (and its neighbourhood to a lesser degree) to **closely resemble/match inputs**

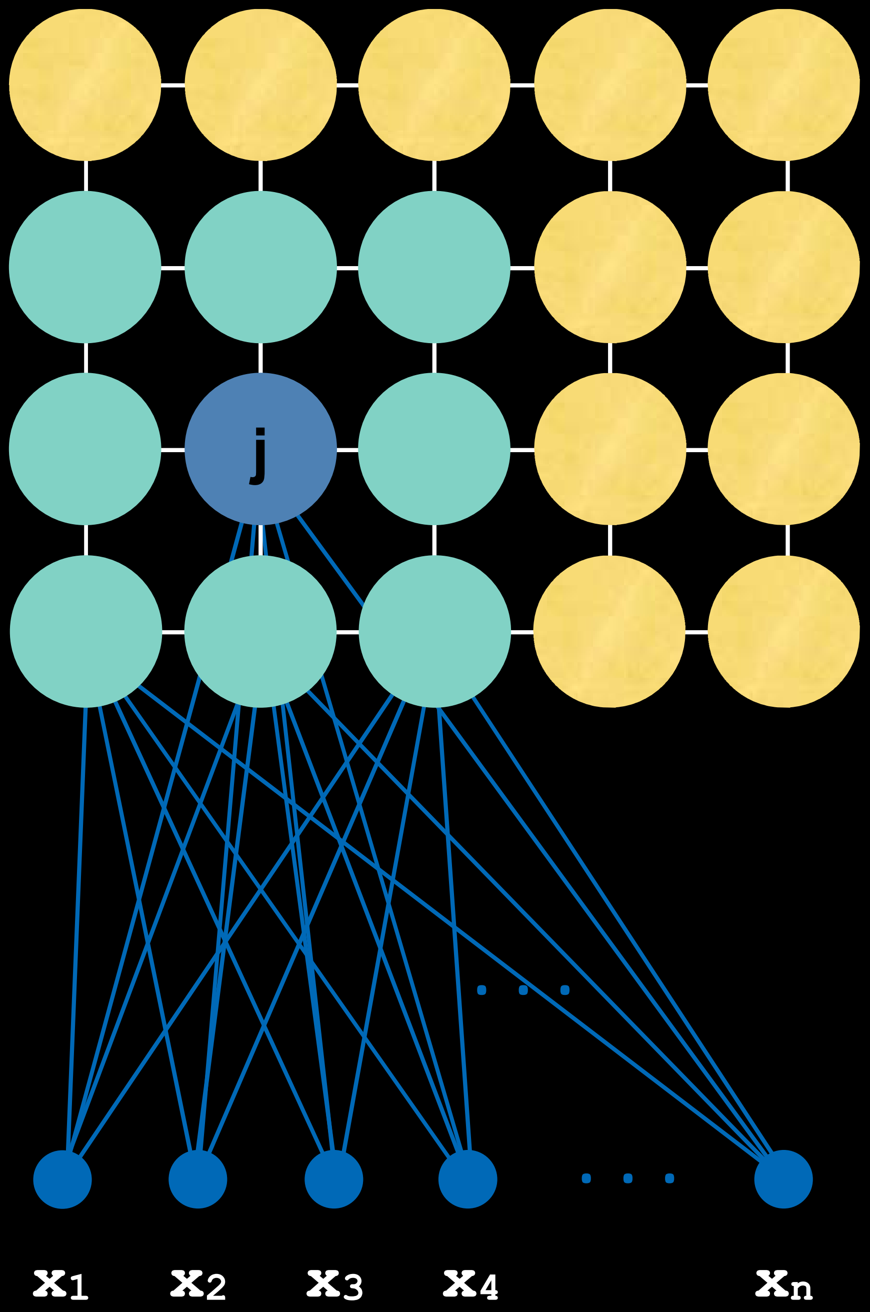
...and so on for all neighbouring nodes...

adapting weights?

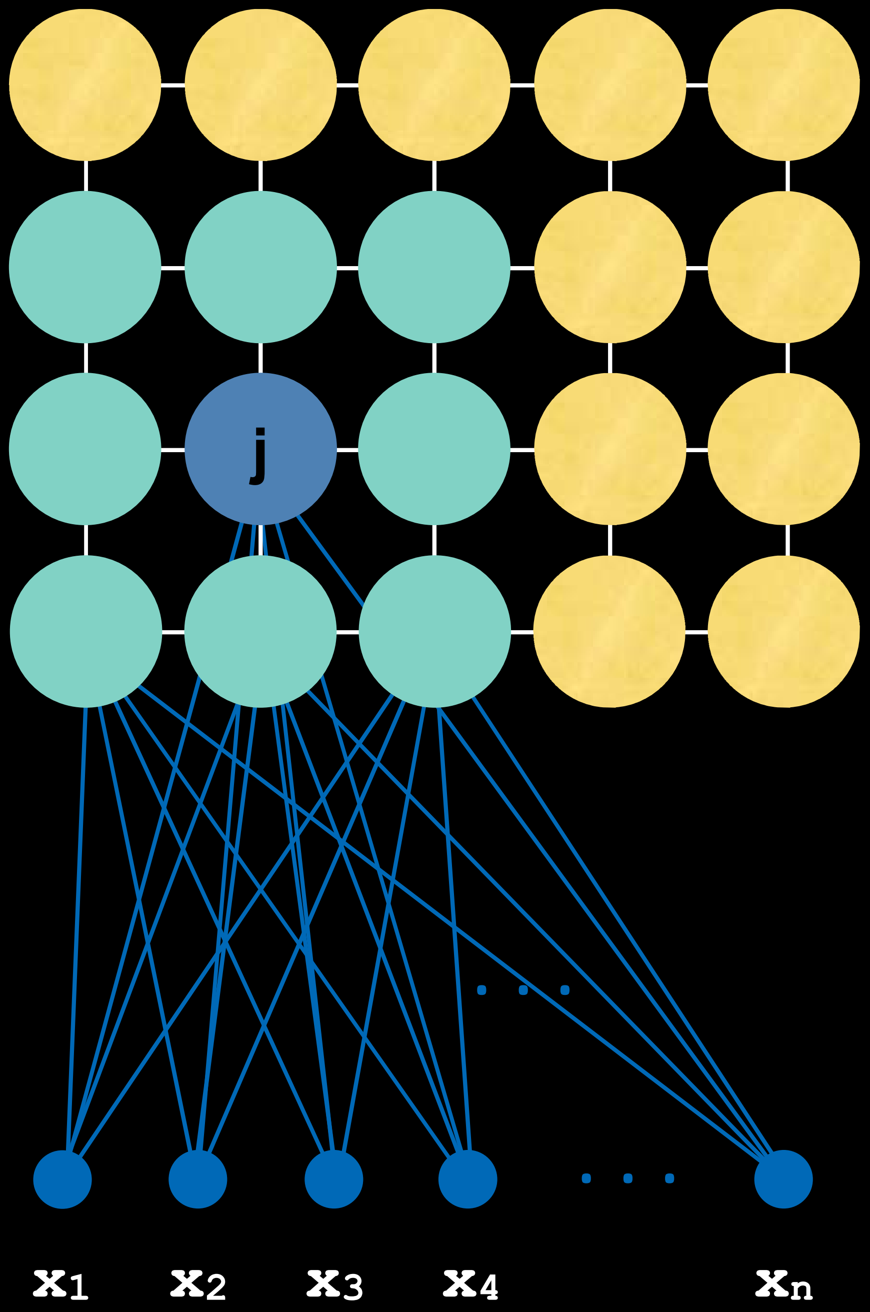


$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta N(i,j)(\mathbf{x} - \mathbf{w}_i)$$

...and so on with N(i,j) deciding how much to adapt a neighbour’s weight vector

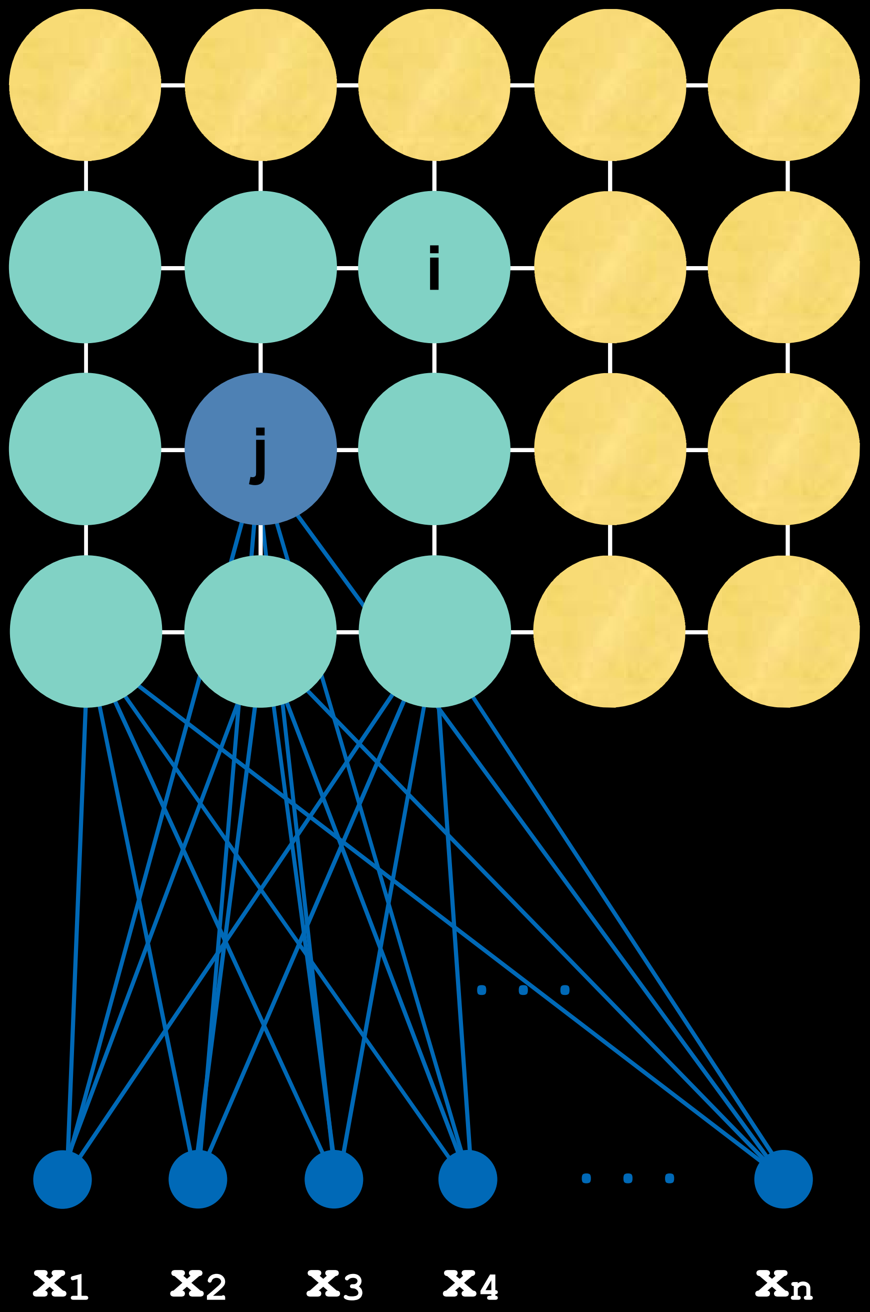


N(i,j) is the neighbourhood function

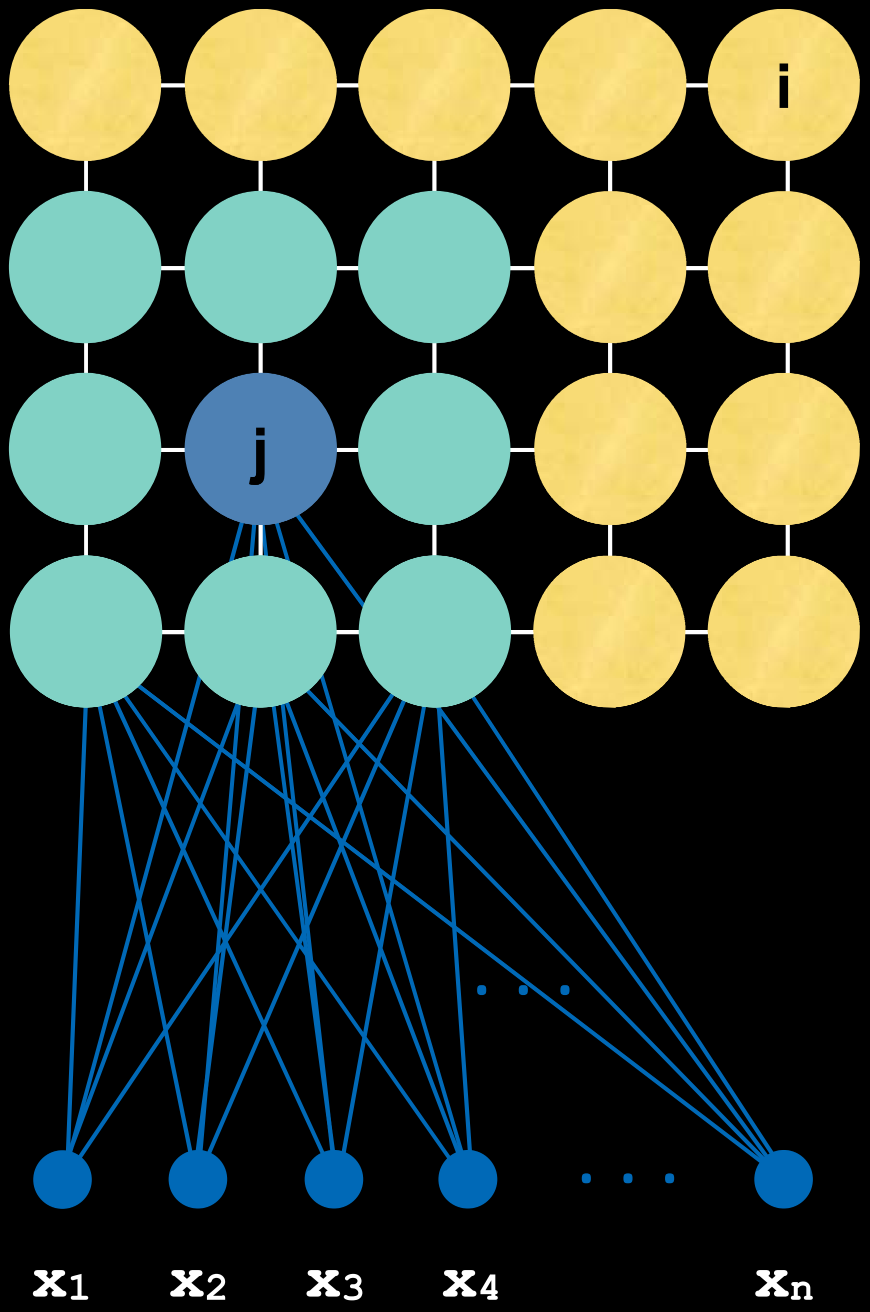


N(i,j) tells how close a neuron i is from the winning neuron j

the closer i is from j **on the lattice**, the higher is N(i,j)



N(i,j) will be rather high for this neuron!

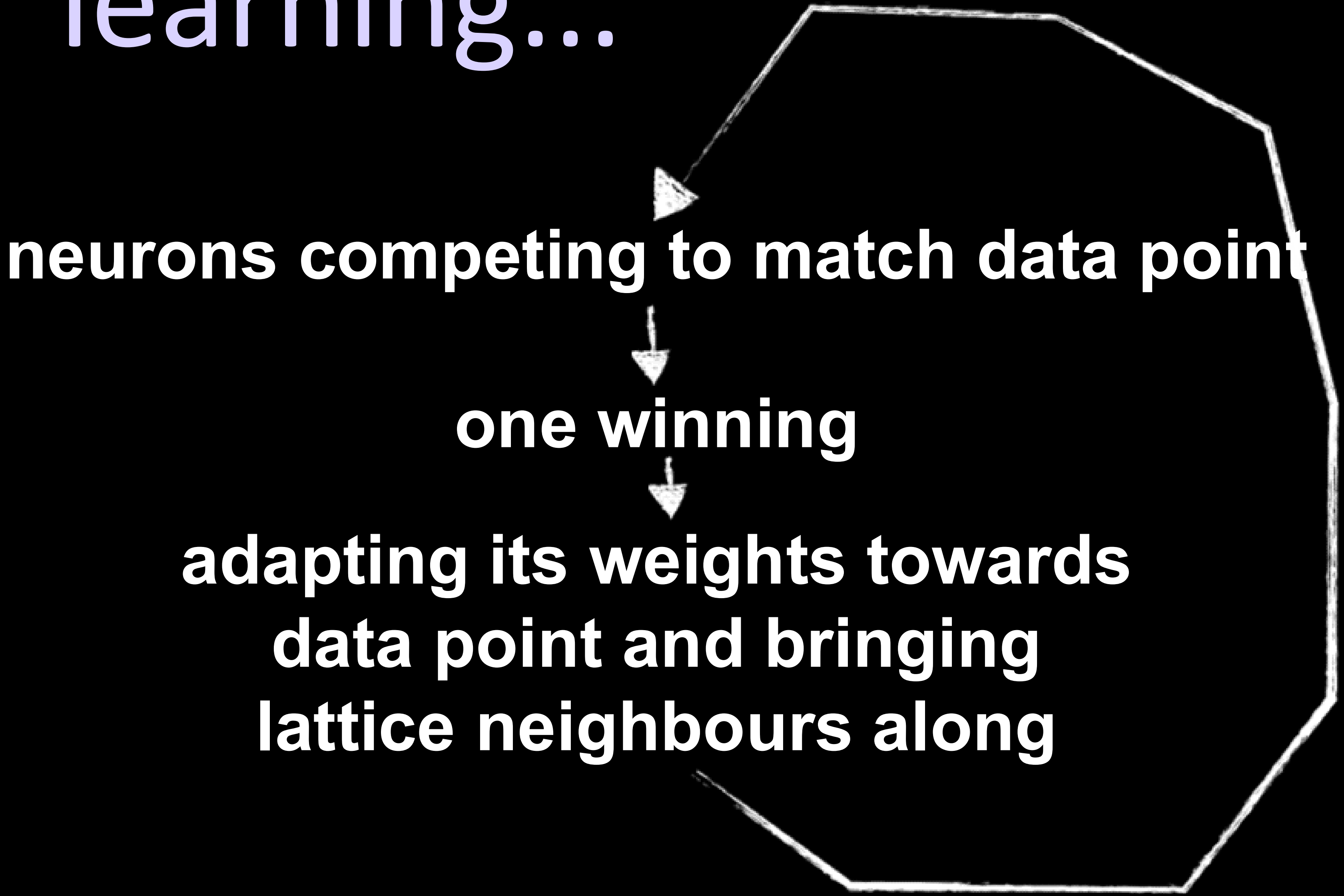


but not as high
for this

so, **update of weight vector** of this neuron will be **smaller**

in other words, this **neuron will not be moved as much towards the input**, as compared to neurons closer to j

summarising learning...



neurons competing to match data point

one winning

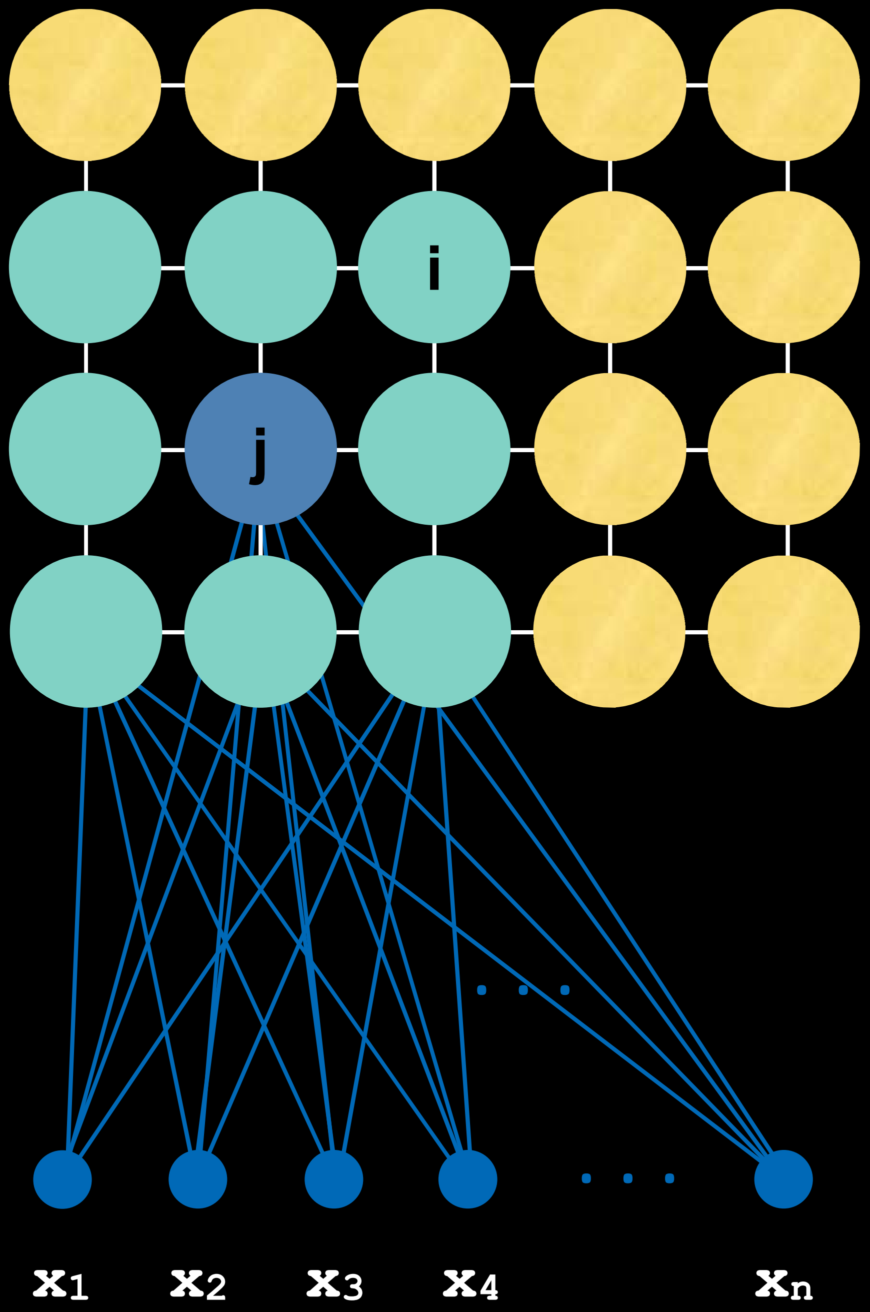
adapting its weights towards data point and bringing lattice neighbours along

with such learning...

- we end up finding weight vectors for all neurons in such a way that **adjacent neurons will have similar weight vectors**!

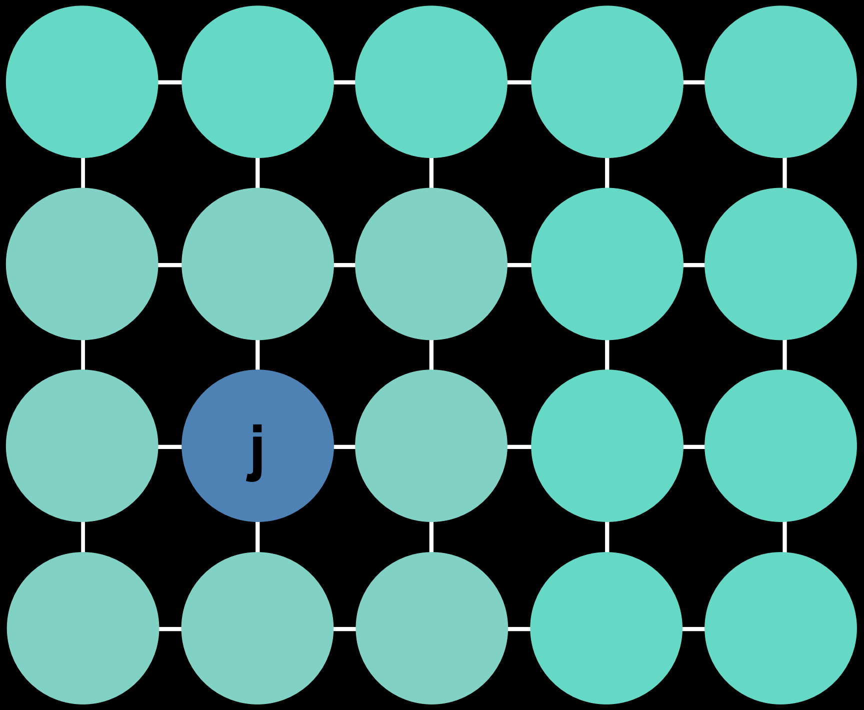
- for any input vector, the output of the network will be the neuron whose weight vector best matches the input vector

- so, each (**weight vector** of a) **neuron is the center of the cluster containing all input data points mapped to this neuron**



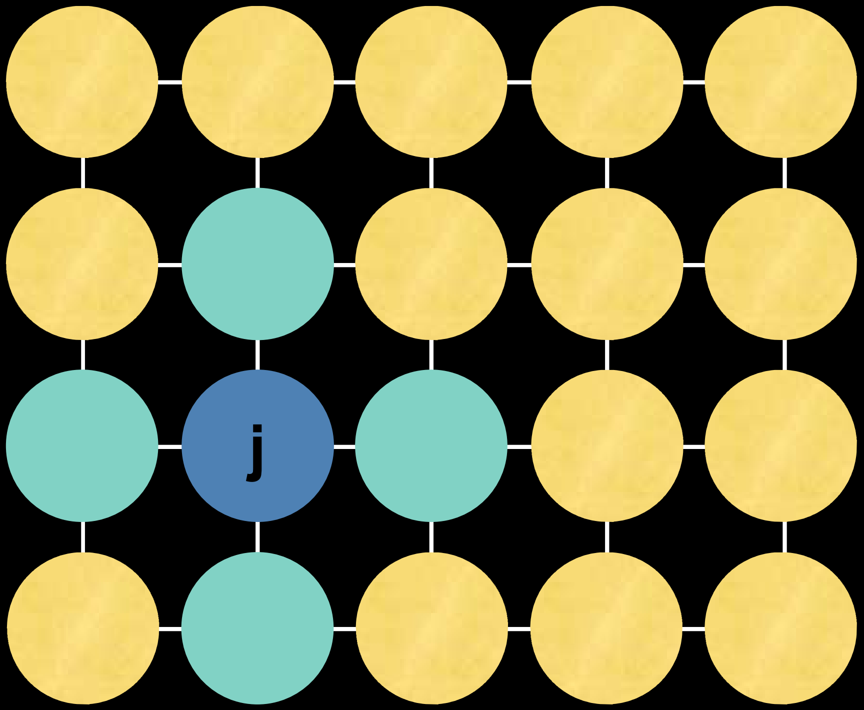
N(i,j) is such that the neighbourhood of a winning neuron reduces with time as the learning proceeds

the learning rate reduces with time as well



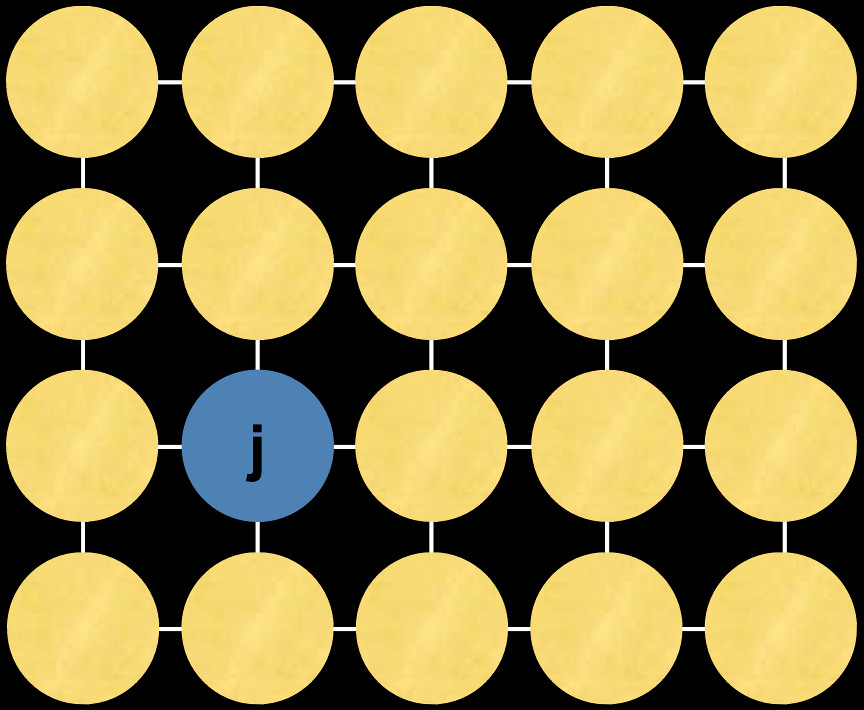
at the beginning of learning the entire lattice could be the neighbourhood of neuron j

weight update for all neurons will happen in this situation



at some point later, this could be the neighbourhood of j

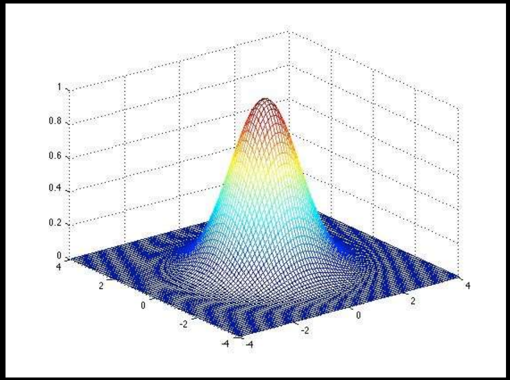
weight update for only the 4 neurons and j will happen



much further on...

weight update for only j
will happen

typically, N(i,j) is a gaussian function



three essential processes

- **competition** - finding the best matching unit/winner, given an input vector
- **cooperation** - neurons topologically close to winner get to be part of the win, so as to become sensitive to inputs similar to this input vector
- **weight adaptation** - is how the winner and neighbour's weights move towards and represent similar input vectors, which are clustered under them

size of the network?

- we determine the size
- big network?
  - each neuron represents each input vector!
  - not much generalisation!
- small network?
  - too much generalisation!
  - no differentiation!
- try different sizes and pick the best...

performance measures?

- **quantization error**: average distance between each input vector and respective winning neuron

$$\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{w}_{winner}\|$$

- **topographic error**: proportion of input vectors for which winning and second place neuron are not adjacent in the lattice

$$\frac{1}{N}\sum_{i=1}^{N} adjacent(winner_i, second_i)$$

self-organising?

- **global ordering** from **local interactions**
  - each neuron interacts only with its neighbours via N(i,j)
  - but the network ends up clustering and preserving topological relationships in data

visualising the resulting SOM?

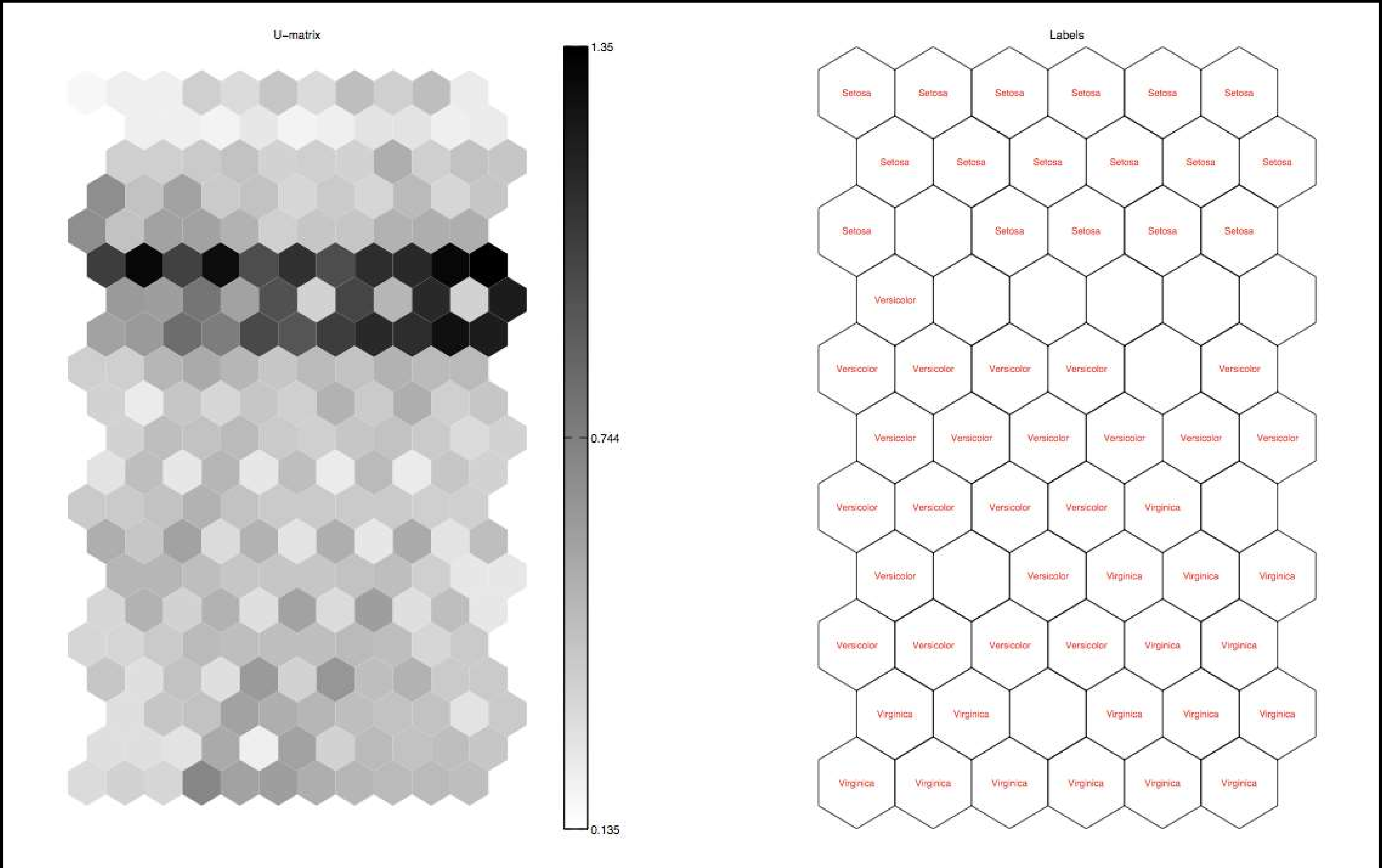
- once the network organises itself over data, how do we visualise it?
  - neurons have weights of input vector dimensions!
  - how to see the discovered similarities/dissimilarities in data in the map space?
  - **U-matrix (unified distance matrix)** is one way

e.g. from SOM toolbox

http://www.cis.hut.fi/somtoolbox/

weight distances between the adjacent neurons are calculated and shown in respective shades/heatmap

neurons with labels



advantages?

- good for **visualisation and interpretability**
- good for **classification** problems
- high **sensitivity to frequent/relevant inputs**
- **new ways** of associating related data

disadvantages?

- system is a **black box**
- a **large training set** may be required
- for large problems, **training can be lengthy**

SOM Toolbox with demo code: http://www.cis.hut.fi/somtoolbox/