Natural Computation Methods in Machine Learning (NCML)

Lecture 10: Growing Neural Gas

(and a recap of CL and SOFM)

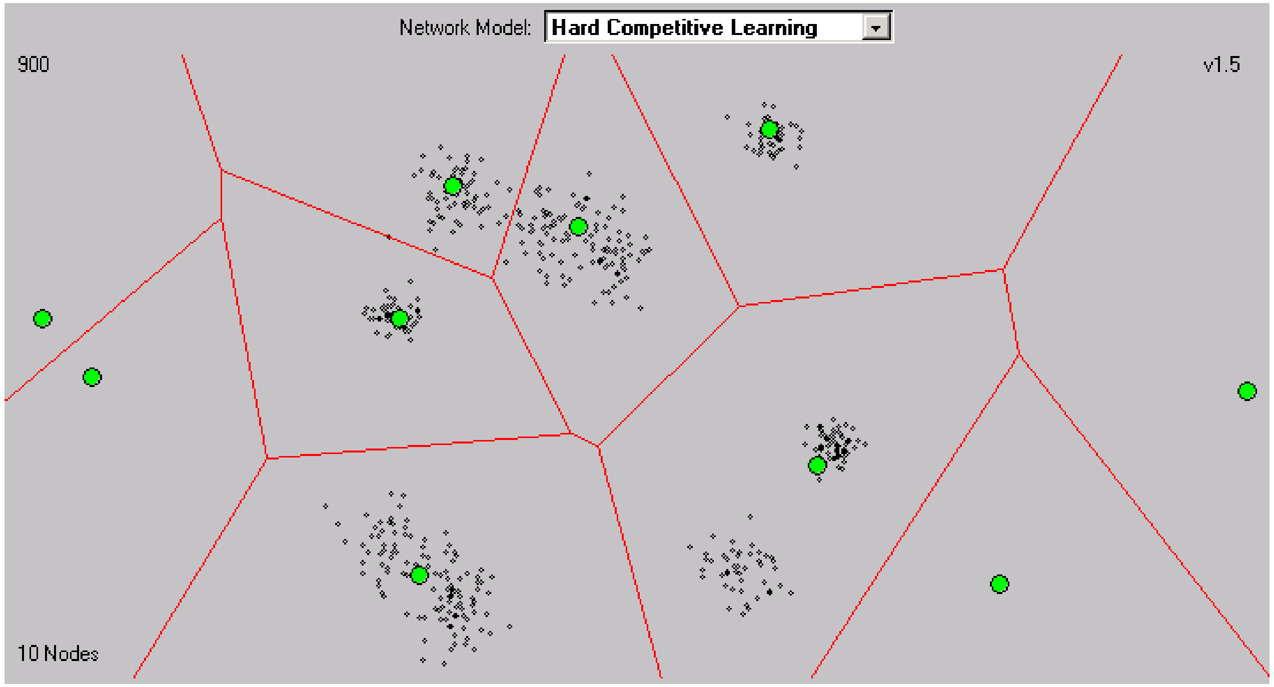
CL and SOFM, revisited

- A node's *position* (in the input space) is its weight vector
  - Nodes move around in the same space as the data
- The purpose of unsupervised learning is (usually) to cluster for classification, or for modelling distributions
  - We want several nodes in high density areas, fewer in low density areas
  - i.e. we want node distributions which follow the data distributions

Competitive Learning

Effects of bad initialization

- Bad utilization if we randomize initial positions

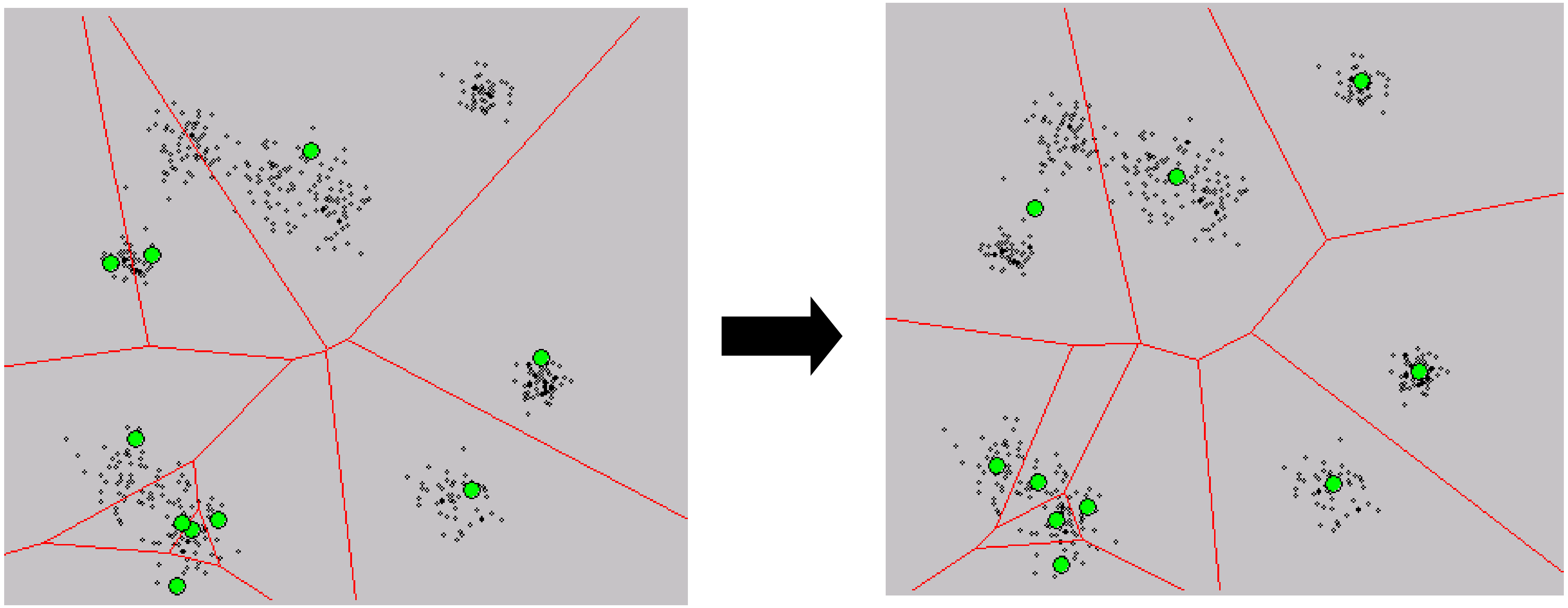


Some nodes (green) are unused – they will never move since their Voronoi regions are empty

Competitive Learning

Effects of bad initialization

- Better if initialized from the data instead
- Still not perfect though (unfair distribution of nodes)
  - Some nodes seem to need help to cover their data clusters

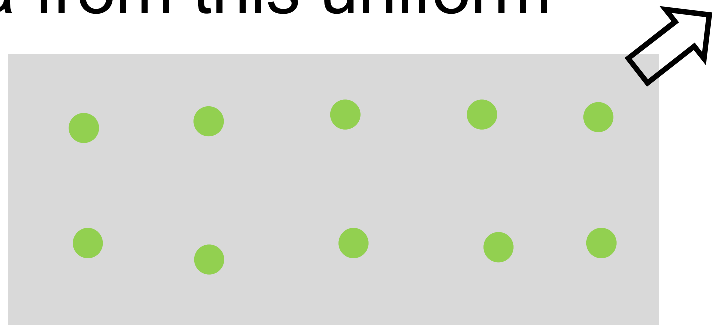


Q: How can we detect, automatically, that a node 'needs help'?
A: It will move around more than the others

Competitive Learning

Problems with non-stationary distributions

- What should we expect if we run competitive learning with 10 nodes on data from this uniform distribution?
  - They should spread out approximately uniformly

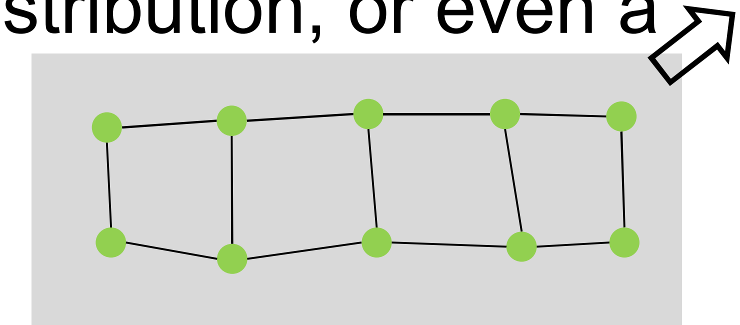


- What if the square moves, very slowly?
  - Works only if it moves very slowly, leaving no nodes behind (without any data in their Voronoi regions)
- What if it moves faster, or jumps?
  - The Winner-takes-all scenario strikes again!
- What if this had been a Self-Organizing Feature Map?

Self-Organizing Feature Maps

Work better for non-stationary distributions

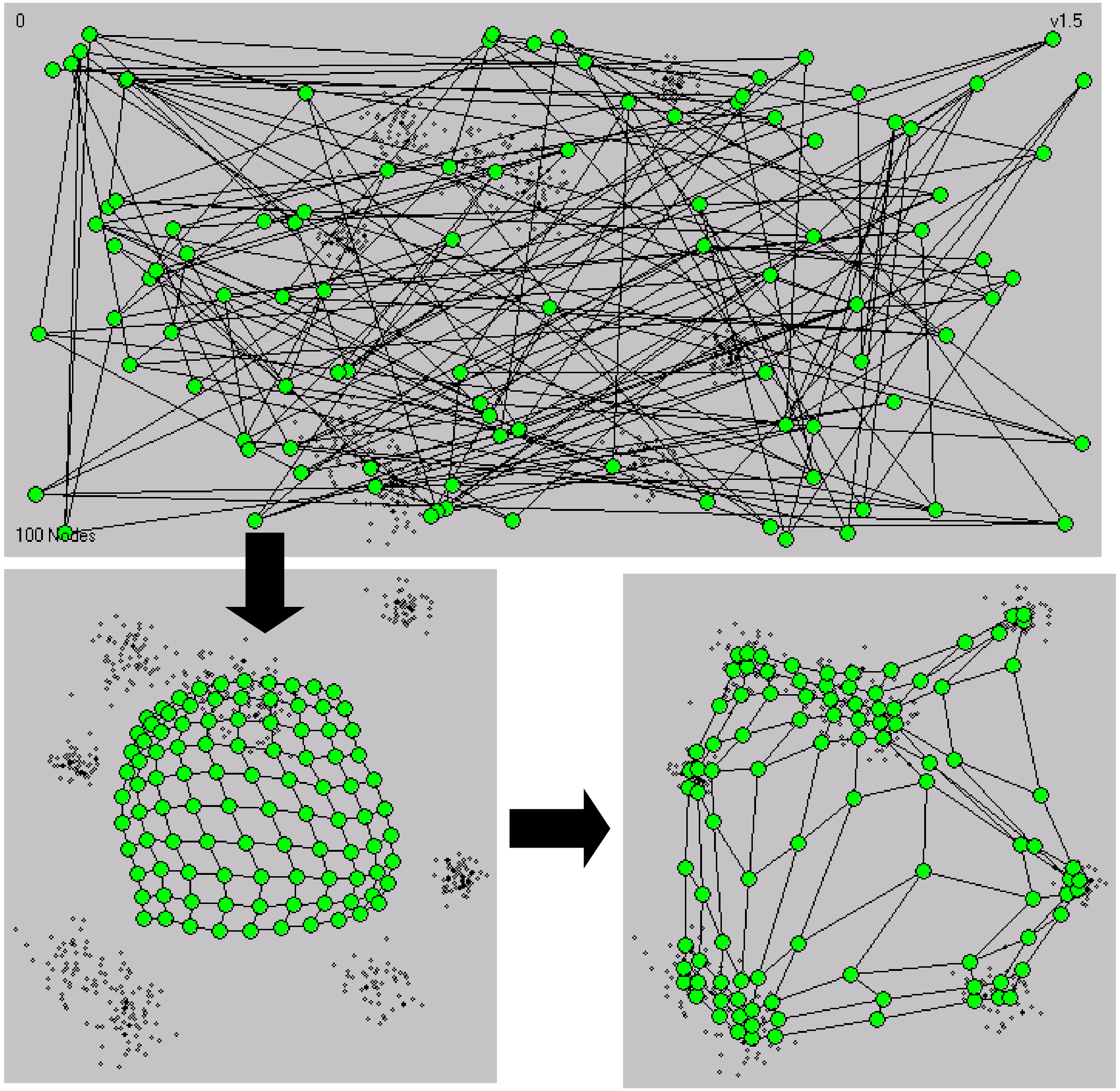
- SOFM will follow a moving distribution, or even a jumping one
  - since winning nodes drag their neighbours along with them



- But, over time, it will gradually lose this ability!
  - due to the decaying parameters
  - (learning rate and neighbourhood function widths)
- Decaying parameters → we impose a time limit
  - for how long do we want to be plastic? (able to adapt)
- *Can we avoid decaying parameters?*

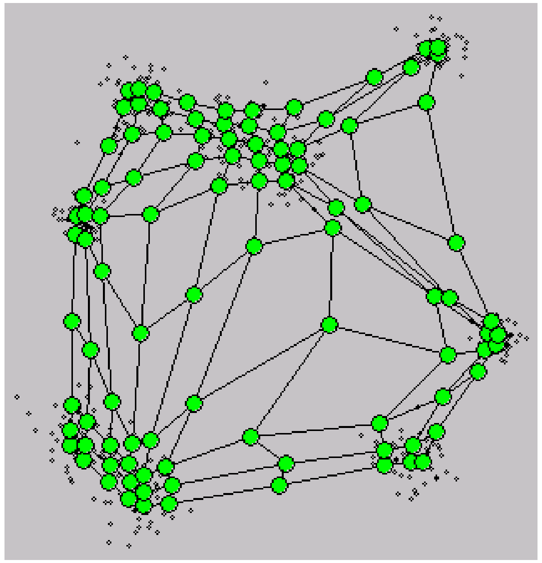
Self-Organizing Feature Maps

- It's actually OK to randomize initial positions
  - Winners will drag their neighbours along with them
- Nodes close to each other, in the map, will very quickly also get similar weight vectors
  - forming a 'fishing net', which then stretches out, trying to fit the data



Self-Organizing Feature Maps

- In SOFM, the *neighbourhood graph* is fixed (a grid)
- In this example, the 'fishing net' is very stretched in the middle
  - some under-utilized nodes there,
  - prevented from moving into a data cluster, by other neighbours pulling in other directions



- There will always be nodes which come in the middle of such conflicts
- Should they still be neighbours?
- Can we make the neighbourhood graph dynamic?
  - Create and cut edges as needed
  - like rubber bands, which snap if too stretched, or too old?

Growing Neural Gas

Fritzke, 1995

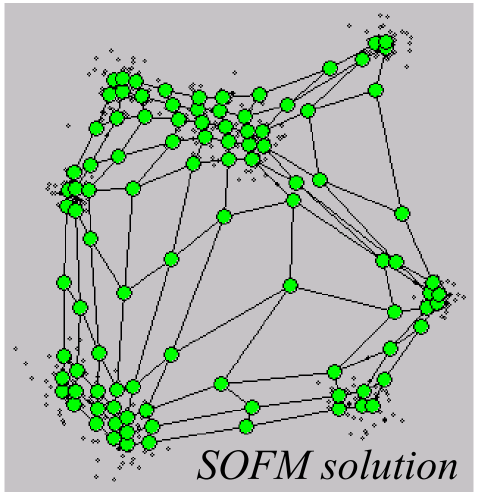
- Unsupervised growing algorithm for clustering
- Dynamic size – a growing/shrinking algorithm
  - though it grows much faster than it shrinks
- Dynamic neighbourhood – who is neighbour to whom is not fixed
  - Defined by a graph (as the grid lines in SOFM above)
- All parameters are constants!
  - GNG will not 'freeze' after a while, as SOFM does
  - Great for on-line learning and moving targets!
  - GNG will even follow jumping targets

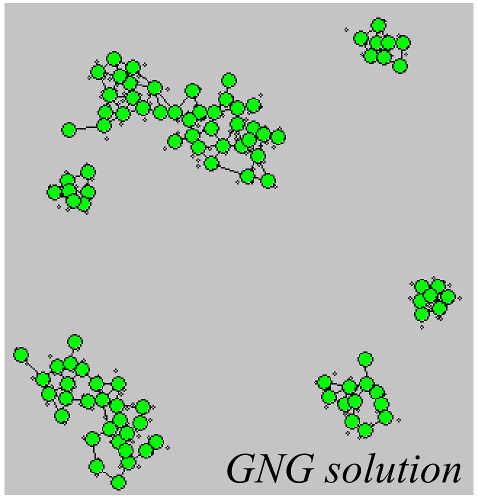
Growing Neural Gas

Implementation considerations

We must decide ...

- Which node(s) to move, given an input
  - and by how much
- How to define and update the neighbourhood graph
  - which is no longer a fixed grid
- How to grow
  - when and where to insert new nodes


SOFM solution


GNG solution

Growing Neural Gas

Implementation considerations

Which node(s) to move, given an input

- Move not only the winner (k), but also its (current) neighbours
- The winner should move with a much greater gain factor than its neighbours:

$$\Delta\overline{w}_k = \epsilon_k(\overline{x} - \overline{w}_k) \qquad \epsilon_k \gg \epsilon_n$$
$$\Delta\overline{w}_n = \epsilon_n(\overline{x} - \overline{w}_n)$$

- where k is the winner, n is a neighbour, $\overline{x}$ is the input vector and $\overline{w}$ is a node position (its weight vector)
- Both gain factors, ϵ_k and ϵ_n, are constants
- All neighbours use the same step length

  (n in ϵ_n is a name, not an index)

Growing Neural Gas

Implementation considerations

Neighbourhood (and removal of nodes)

- All current neighbours are connected in a graph
- Each edge in the graph has an associated *age*
- For each input vector
  - find the closest node (k, the winner), and the second closest (r)
  - If k and r are not already connected (i.e. neighbours), connect them with a new edge
  - Set the age of the edge between k and r to 0 (zero)
    - The age of all other edges from k is incremented by one
    - If any edge becomes too old ($> a_{max}$), remove it
    - If any node loses its last edge, remove that node as well

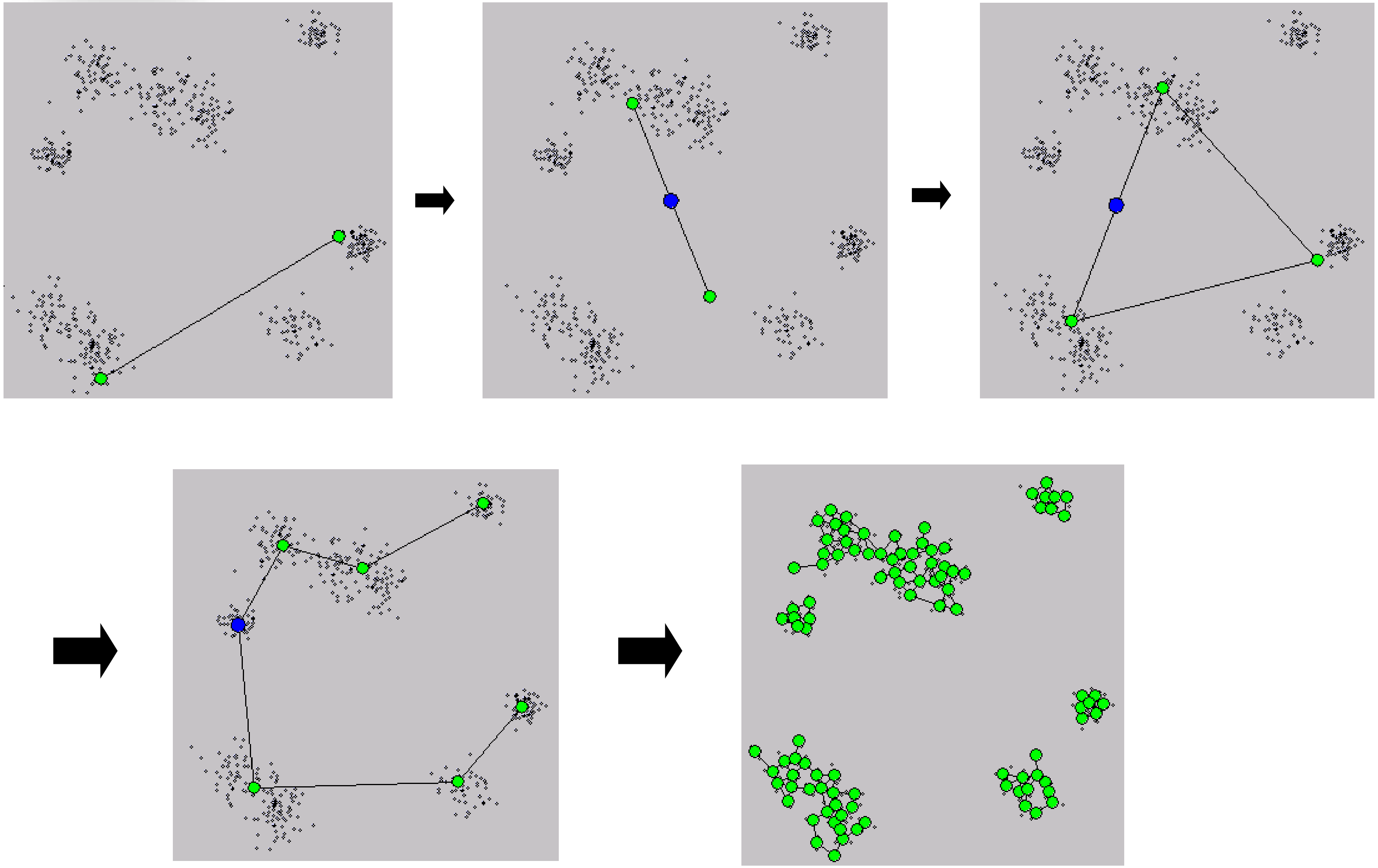
Growing Neural Gas

Implementation considerations

Growth

- Every time a winner, k, is found, add the distance (from the input) to a local error variable
  - The error is proportional to the accumulated distance this node as moved, *as a winner*
- At fixed time intervals, insert a new node where it is most likely needed:
  - halfway between the node with the largest error, and the node among its current neighbours with the largest error
- The error of a node is decayed over time
  - this is not a decaying parameter, though

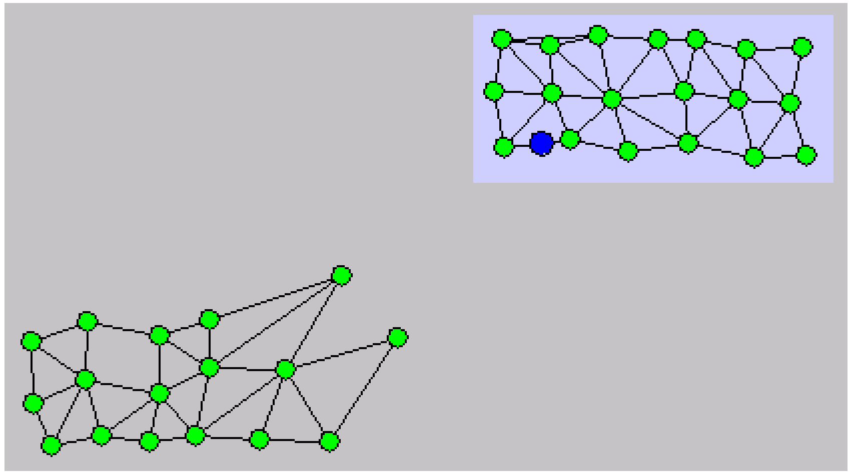
GNG in action



Growing Neural Gas

Possible problem: dead units

- There is only one way for an edge to get 'younger'
  - when the two nodes it interconnects are the two closest to the input, the age is reset to 0
- If one of the two nodes wins, but the other one is not the runner-up, then, and only then, the edge ages
- If *neither* of the two nodes win, the edge does not age!



The input distribution has jumped from the lower left to the upper right corner, leaving a set of nodes in the lower left which will never be used (unless the distribution jumps back again). If they are never used, they will not be removed. Not necessarily a problem though (since it might jump back again).

Growing Neural Gas

Possible problem: dead units

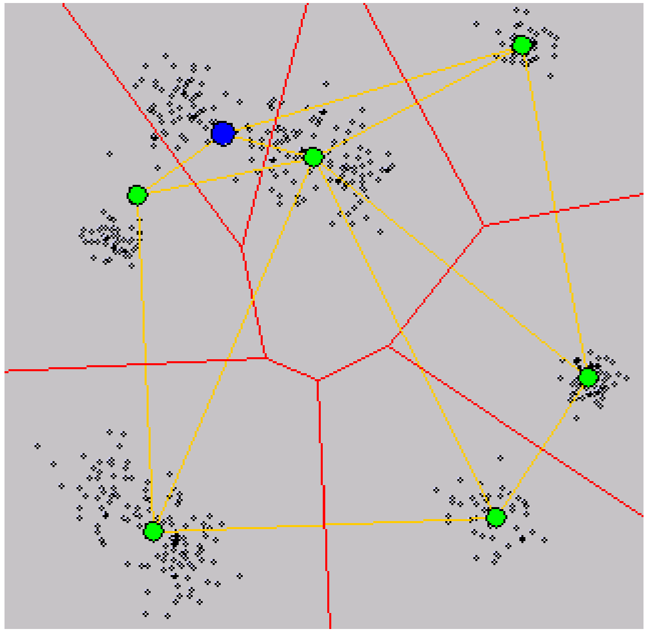
- Common extension: GNG-U
  - Removes nodes with low *utility*, based on frequency of winning and closeness to other nodes
- GNG forms a subset of a *Delaunay triangulation*

- GNG is used mostly for modelling distributions in image analysis
- Can also be used to train the hidden layer in RBF networks (lecture 15)
  - ➔ Automatic sizing of that hidden layer

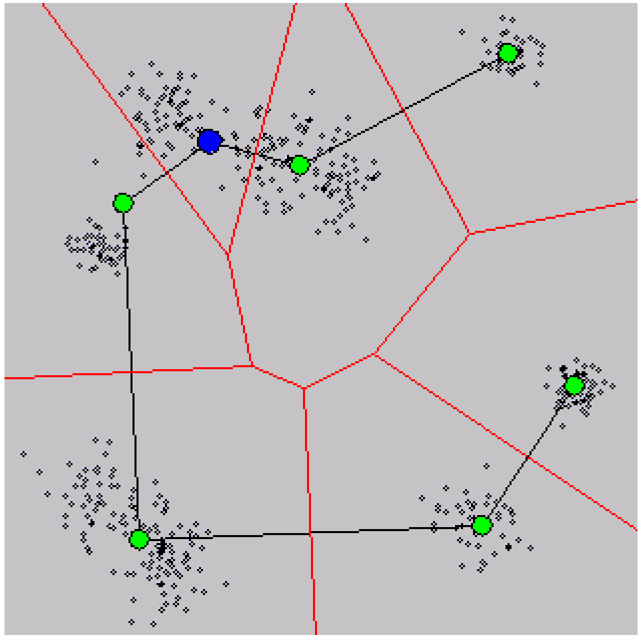
- See the algorithm in Fritzkes paper for details

Delaunay triangulation

Connect the codebook vectors in all adjacent Voronoi regions



Voronoi regions (red) and Delaunay triangulation (yellow)

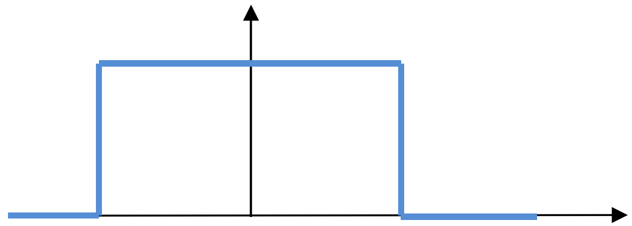


The neighbourhood graph in GNG is a subset of the Delaunay triangulation

SOFM in Matlab

Lab 3

- The grid is hexagonal, not square
- Epoch learning, instead of pattern learning
- The neighbourhood function is a tophat function!



- Step length, η, is 1!
  - This means that all neighbours in a radius around the winner, are moved to the input (not just towards it)!
  - At first glance, this should not work!
  - But in combination with the use of epoch learning, it does! (since weight changes are accumulated over the whole training set, before actually applied)