13 December, 2016

Neural Networks

Course 10: Self Organizing Maps

Overview

- What is a SOM?
- How training works?
- SOM implementation
- SOM on MNIST digits

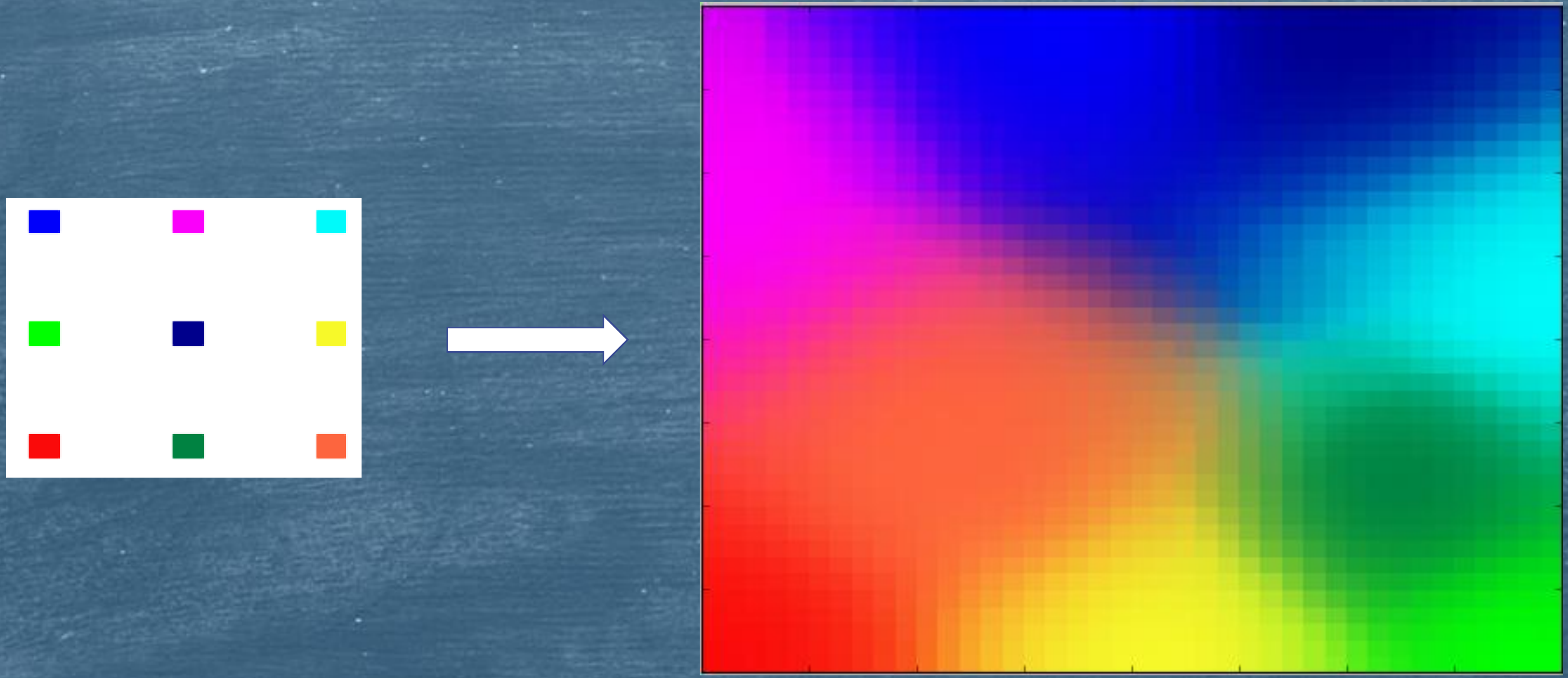
What is a SOM

What is a SOM?

- A self organizing map is a type of artificial neural network (ANN) that is trained to produce a low-dimensional representation of the input space of the training samples
- Also known as Kohonen map or networks from the name of Teuvo Kohonen, who Invented in them in 1980s
- The main feature of Kohonen map is that it stores information in such a way that any topological relationship within the training set is maintained.
- Because of this feature, Kohonen map is used to visualize the relations within the training data. (The dimensions usually used are 2 or 3)

What is a SOM?

- A classic example is mapping of colors (which has three dimensions) to a SOM of 2 dimensions:

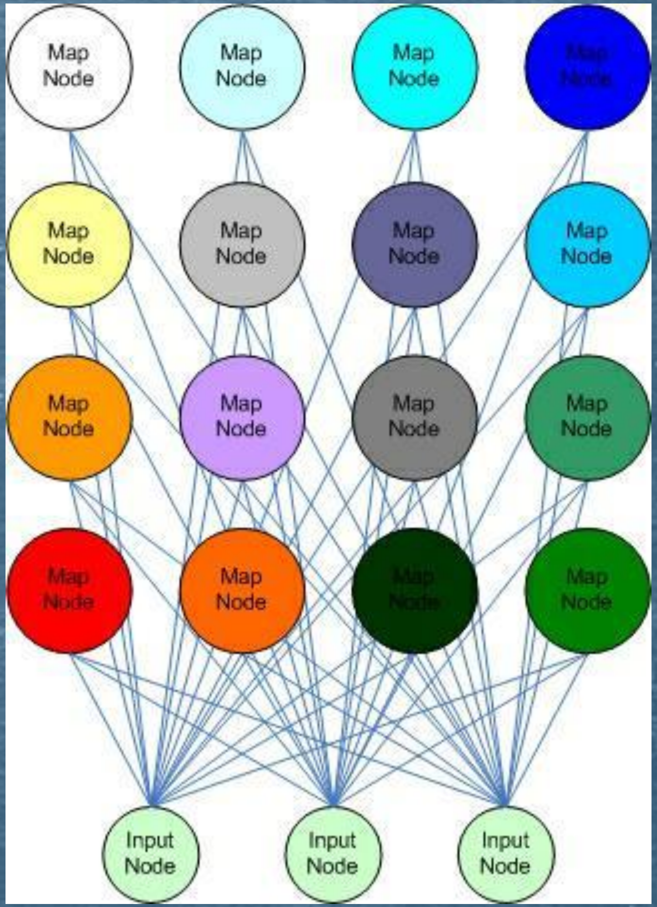


What is a SOM?

- Observations:
  - Different colors are clustered in different regions
  - Adjacent colors are still adjacent in a SOM:
    - i.e. the darker green is close to the light green
  - Opposite colors are in opposite corners:
    - i.e. red, green and blue are located in different regions
  - Colors that are made of basic colors are situated somewhere in the middle:
    - Yellow is in the middle of red and green
    - Violet is in the middle of red and blue
    - Light blue is in the middle of green and blue

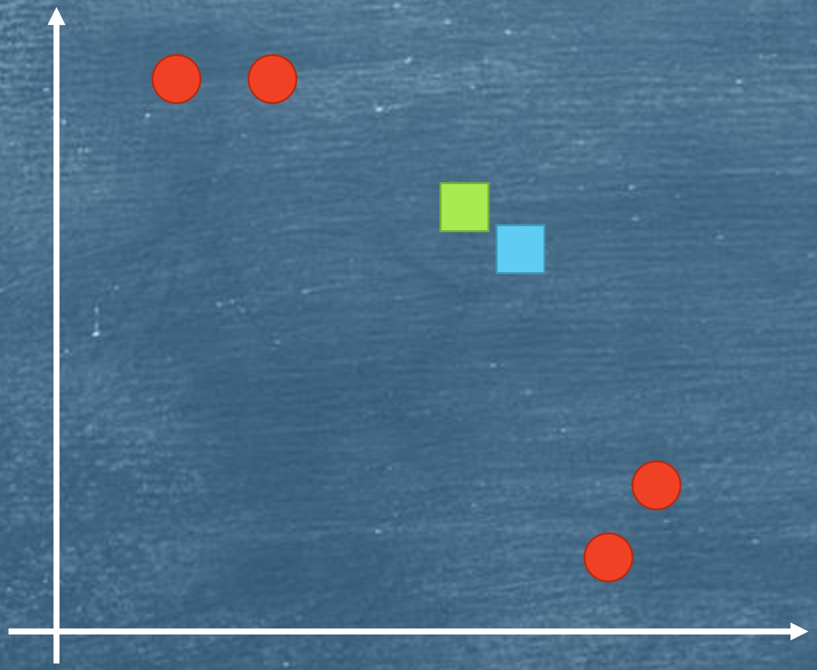
What is a SOM?

- Network Architecture:
  - The map consists of a matrix of nodes, each one having its own coordinates (x,y for 2 dimensions, x,y,z for 3 dimensions)
  - Each node from the Map connects with every feature from the input. Each connection has a weight
  - There are no connections between nodes in the map
  - In the previous example, the map consisted of 40x40 nodes



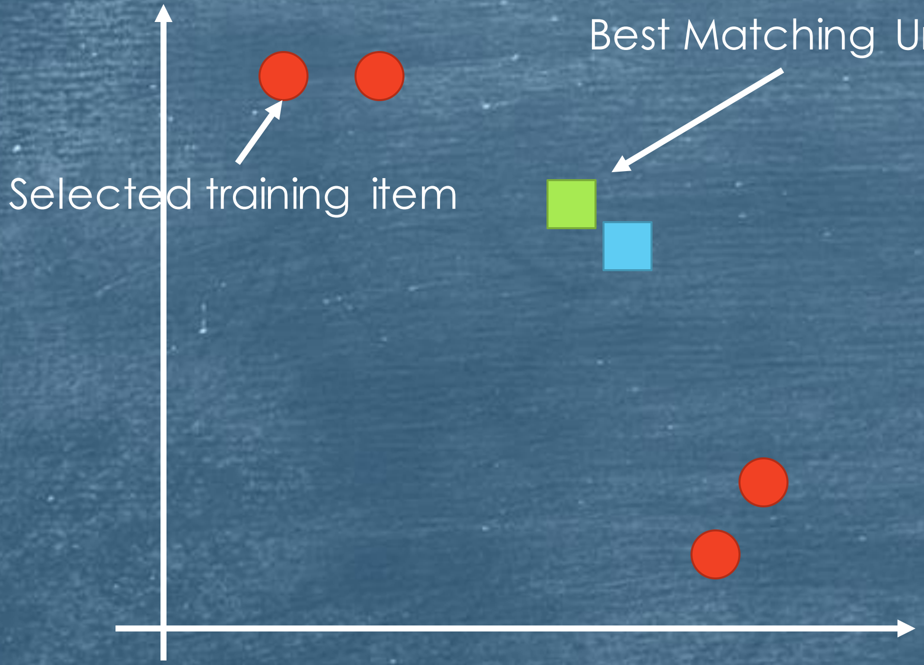
How training works?

How training works?



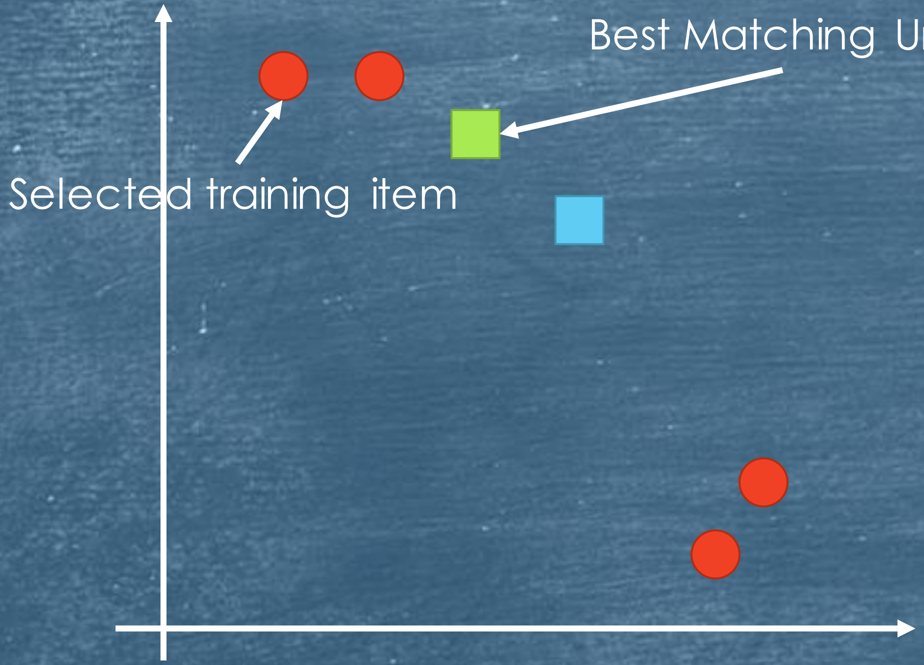
- Consider the case on the left:
  - Each circle represents a training item, identified by the features :x,y in this case.
  - Each square represents a node from the SOM represented by the weights with the input.
  - If we were to cluster colors, the circles and squares would have had three coordinates (r,g,b for circles) (weights for r,g,b for squares)

How training works?



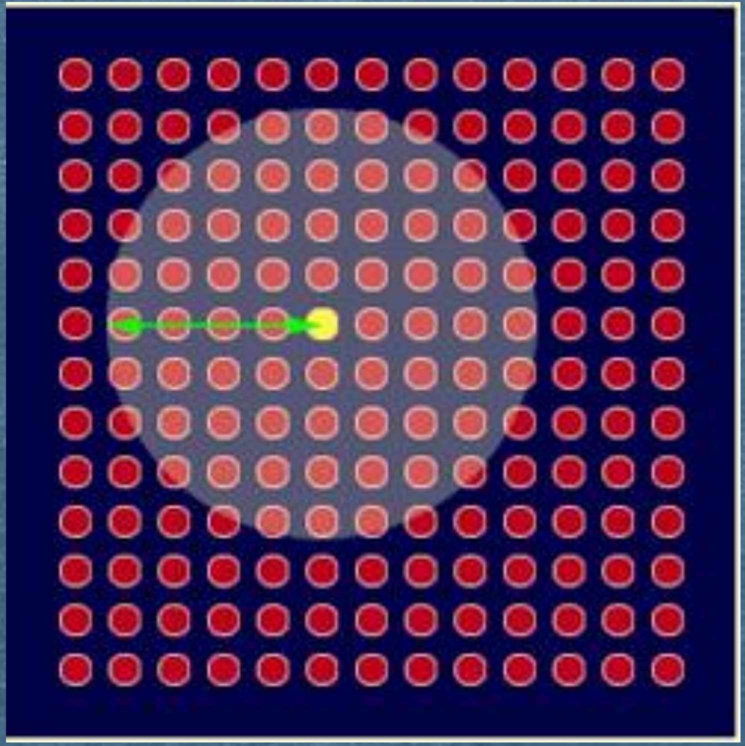
- On each iteration:
  - A training item will be selected
  - A distance will be computed to every SOM unit and a best matching unit will be selected. (the closest one)

How training works?



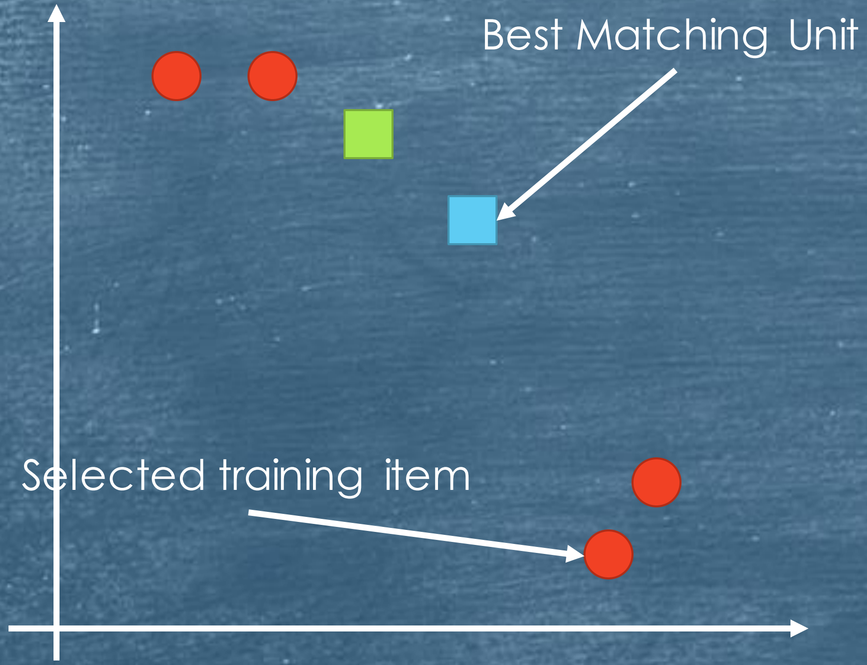
- On each iteration:
  - The best Matching unit will be made to look more similar to training example
  - However, not only the BMU is affected. The units that are in the proximity of the BMU (in the map) are also affected .
  - The further they are, the less they are affected

How training works?



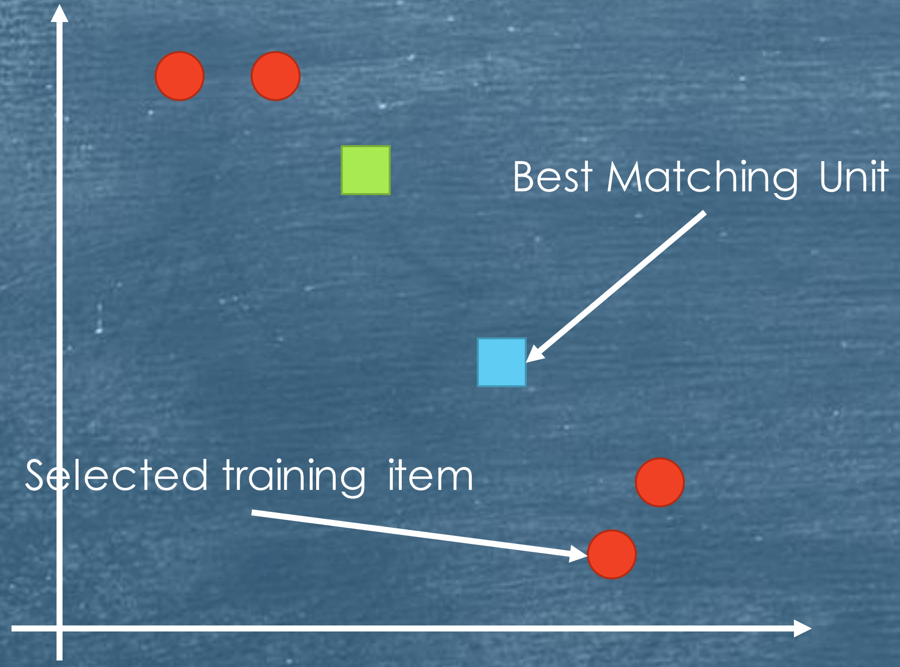
- On each iteration:
  - Consider that the yellow circle is chosen to be the BMU (here, the MAP has (13x13 units))
  - Its weights will be adjusted to be more similar to the training item.
  - However, all the units within a radius will be made more similar to the training item. The further they are from the BMU, the less they'll be affected
  - The radius shrinks with each iteration until it affects only the BMU

How training works?



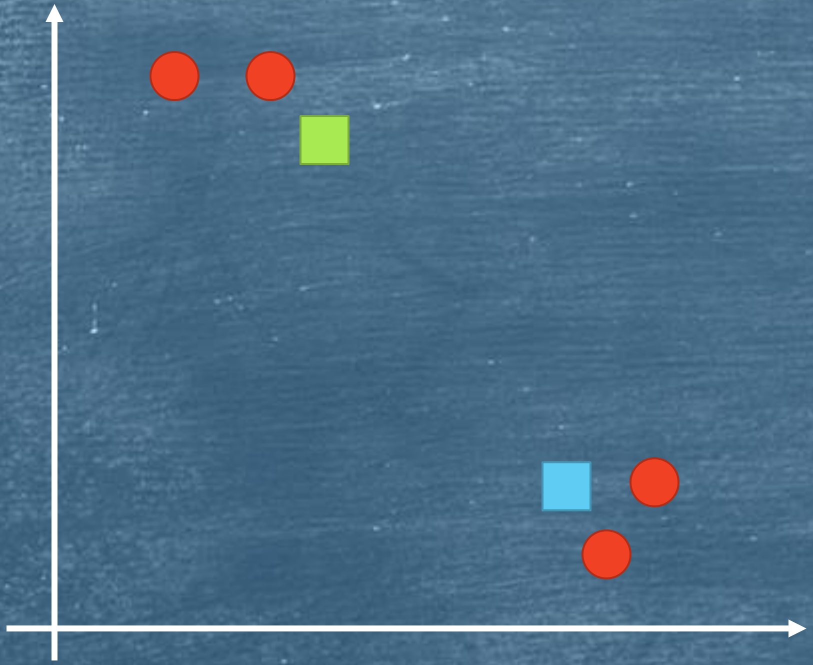
- On the next iteration:
  - Another item will be selected
  - The Best Matching Unit will be computed again (which might be another node)
  - The BMU will be made to look simmilar to the training input and will affect adjacent units. However, the number of units it affects decreases. As well as the magnitude

How training works?



- On the next iteration:
  - Another item will be selected
  - The Best Matching Unit will be computed again (which might be another node)
  - The BMU will be made to look simmilar to the training input and will affect adjacent units. However, the number of units it affects decreases. As well as the magnitude

How training works?



- With each iteration:
  - The units will adapt at a lower rate (to not overshoot)
  - The number of units that get affected by the BMU decreases (to not disturb the units that have already arrived in the right place)
  - The amount that each unit near the BMU gets affected, decreases (presumably, most of the items are already at a good position)
- In the end:

The units would have self-organized, meaning that they will be similar to a subset of the training data

SOM implementation

SOM implementation

- The training consists of the following steps:
  - Initialize the weights for each node
  - On each iteration:
    - Choose a training item from training set
    - Compute the distance to each node and get the closest one: best matching unit (BMU)
    - Compute the radius of the BMU (how many additional nodes it affects)
    - Adjust the nodes from the previous step to be more like the input vector

SOM implementation

- Initialize the weights for each node:
  - Each node has as many weights as the size of the training item
  - So, if there are 40x40 nodes and each input vector is defined by 3 features (r,g,b) then, the network will have 40x40x3 (4800) weights in total
  - Each weight is initialized with a small random values (between 0 and 1)
  - Depending on how the data is stored, it is sometimes necessary to store the coordinates of each node

SOM implementation

- Compute the BMU:
  - Compute the distance between the selected training item and every unit of the SOM. The unit that is closest to the training item is chosen to be the BMU
  - Usually, the Euclidian distance is used:

$Dist = \sqrt{\sum_{i=0}^{i=n}(V_i - W_i)^2}$ where $V\ is\ the\ input\ vector, W\ is\ the\ weight\ vector$

SOM implementation

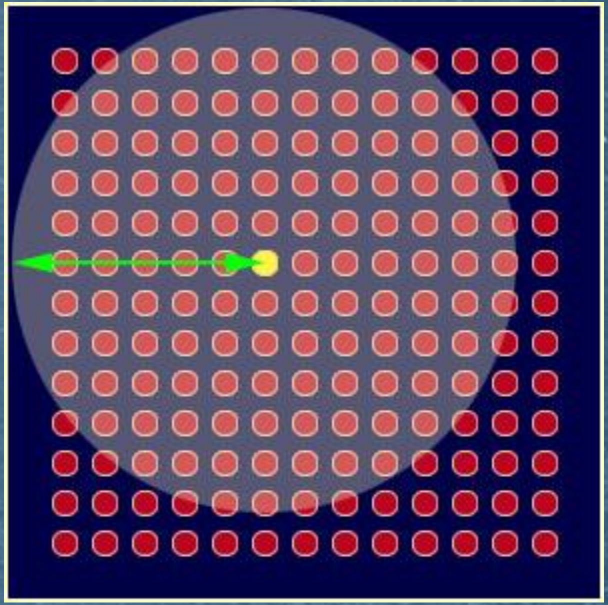
- Compute the radius of BMU:
  - The radius of BMU starts very large and shrinks with each iteration:
  - $\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right)$

    Where:

    $\sigma_0 =$ the radius at the first iteration. Usually this is set to $\max(\frac{SOM\_width}{2}, \frac{SOM\_height}{2})$

    t = iteration number

    $\lambda = \frac{T}{\ln(\sigma_0)}, where\ T = total\ number\ of\ iterations$



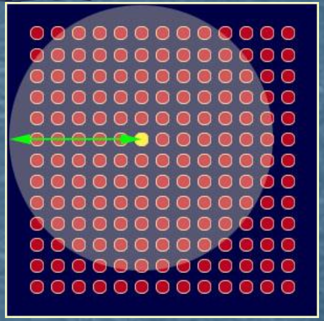
SOM implementation

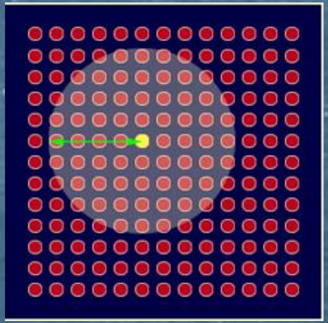
- Compute the radius of BMU:

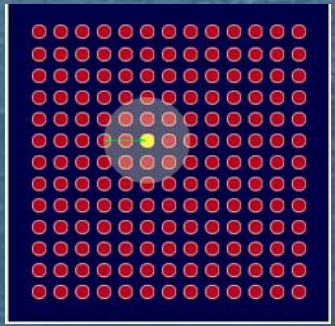
$$\sigma(t) = \sigma_0 e^{\frac{\frac{-t}{T}}{\ln(\sigma_0)}} = \sigma_0 e^{-\frac{t}{T}\ln(\sigma_0)} = \sigma_0\left(e^{\ln(\sigma_0)}\right)^{-\frac{t}{T}}$$

$$\sigma(t) = \sigma_0 \cdot \sigma_0^{-\frac{t}{T}}$$

Observe that when t equals T, $\sigma(t)$ will be 1.
Until then, $\sigma(t)$ will be a fraction of σ_0







SOM implementation

- Adjust the units

Every unit that is within the BMU radius (including the BMU) will be adjusted to be more similar to the training item.

$$W_{t+1}^{i} = W_{t}^{i} + \underbrace{\Phi_t}_{\text{Influence of distance to BMU}} \underbrace{\eta_t}_{\text{Learning rate, adjusted at iteration t}} (V^{i} - W_{t}^{i})$$

SOM implementation

- Adjust the units. The Learning Rate

The learning rate must decrease with each iteration in order to not overshoot the target.

$$\eta_t = \eta_0 * \exp\left(-\frac{t}{T}\right)$$

Where η_0 is the initial learning rate, usually set as 0.1

SOM implementation

- Adjust the units. Influence of distance to BMU

Each unit inside the radius of the BMU will have its weights adjusted

The amount by which is adjusted depends on how far it is from the BMU.

An usual approach is to use the same formula used on the Radial Basis Function Network, that is based on the normal distribution

$$\phi_t = \exp\left(-\frac{dist^2}{2\sigma_t^2}\right)$$

where

$dist$ represents the distance from one unit to the BMU

σ_t represents the radius of the BMU

Questions & Discussion

References

- http://www.ai-junkie.com/ann/som/som1.html
- http://www.brains-n-brawn.com/default.aspx?vDir=aisompic
- https://en.wikipedia.org/wiki/Self-organizing_map