SBE 4032: Advanced Topics in Medical Informatics (2)

موضوعات متقدمة في المعلوماتية الطبية (2)

Lecture #4: Dimensionality Reduction — Linear Methods and Non-Linear Methods

Hisham Abdeltawab, Ph.D.
Assistant Professor,
Cairo University

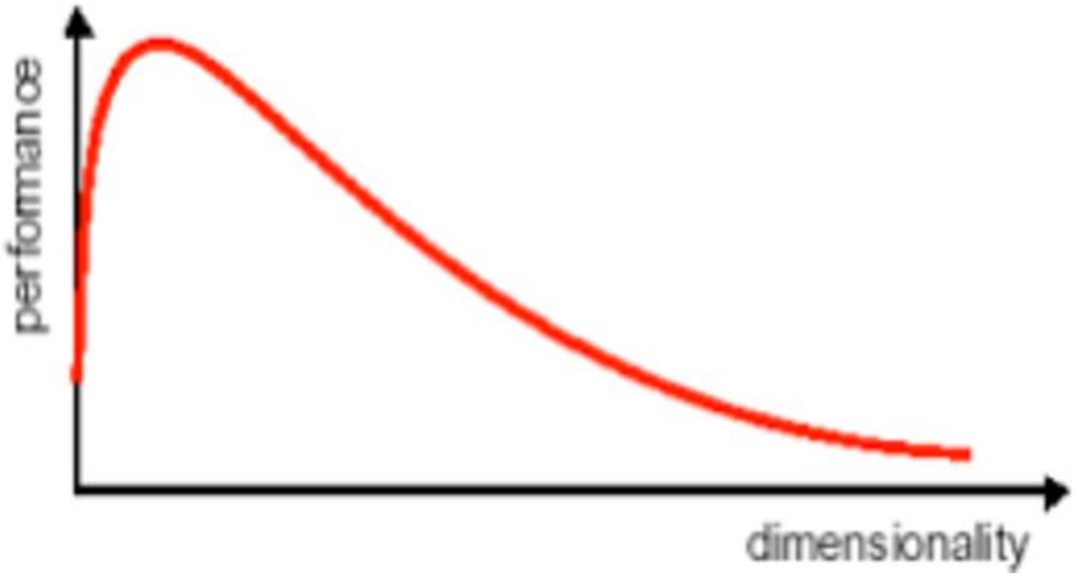
Why Dimensionality Reduction?

Features Matrix

p features (dimensionality)

n observations

0.6	0.3	….	….	….	…	…	0.2



- It increases the performance machine learning models by removing noise and redundancy.

Random Projection: Motivation

- In many modern data analysis situations, the dimensionality of the data (p) is so large that the use of standard linear dimensionality reduction methods based on PCA is either infeasible or intractable.

- Time complexity of PCA: $O(\underbrace{p^2n}_{\text{Covariance}} + \underbrace{p^3}_{\text{Eigen-value decomposition}} + \underbrace{npd}_{\text{Projection}})$ where the data is n x p and the reduced dimension is d.

- Time Complexity of Random Projection: $O(\underbrace{npd}_{\text{Projection}})$

Random Projection

- We can transform the training samples X to a lower dimensional space by a random transform as follows:

$$X_d = \sqrt{\frac{1}{d}} R^T X$$

X_d

Training samples randomly projected into a reduced dimension feature space.
Size = d x n

R

Random matrix and its entries chosen from a standard normal distribution.
Size = p x d

X

Is the original training samples in feature space.

Size = p x n

d

The reduced dimension.

$d < p$
p is very large

Random Projection

- Johnson and Lindenstrauss [1984] show that the entries of R can be chosen from a standard normal distribution.
- They also show that this type of projection ensures (with high probability) that the interpoint distance in the d-dimensional subspace matches approximately those in the original p-dimensional space.

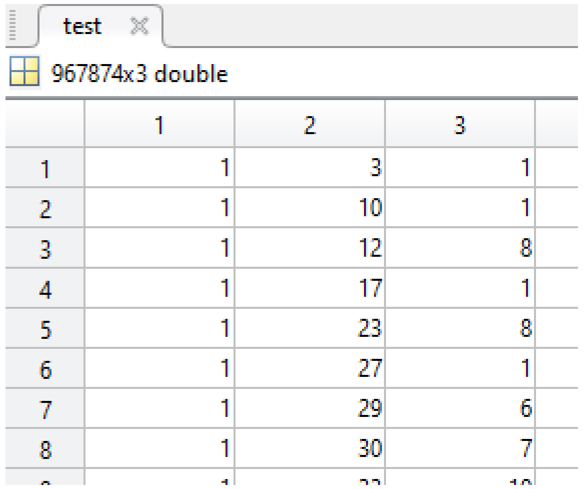
Random Projection: Example

A famous data set for text analysis is the 20 Newsgroup data.

```
% This has been downloaded from
% http://qwone.com/~jason/20Newsgroups/
% Choose the MATLAB/Octave link and
% extract to your current directory.
load('test.data')



% We convert this to a sparse version.
testsp = sparse(test(:,1),...
                test(:,2),test(:,3));
% Determine the size of the data matrix.
[n,p] = size(testsp);
```



test

967874x3 double

	1	2	3
1	1	3	1
2	1	10	1
3	1	12	8
4	1	17	1
5	1	23	8
6	1	27	1
7	1	29	6
8	1	30	7

The size function returns $n = 7505$ documents and $p = 61188$ words. Thus, we are in a very high-dimensional space.

Random Projection: Example

- Thus, we are in a very high-dimensional space. The cosine of the angle between two document vectors is typically used to determine how similar they are.
- Recall that the cosine between two (column) vectors **x** and **y** is given by:

$$\cos(\theta_{\mathbf{x},\mathbf{y}}) = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

We can convert to a distance by taking

$$D_{\cos(\theta_{\mathbf{x},\mathbf{y}})} = 1 - \cos(\theta_{\mathbf{x},\mathbf{y}})$$

Random Projection: Example

```
% Here we calculate the cosine similarity:
% cosine(theta) = (x * y)/(||x||*||y||)
% First, the x*y part of the cosine.
tmp1 = testsp * testsp';


% Here is the ||x||*||y|| part.
tmp2 = sqrt(diag(tmp1))*sqrt((diag(tmp1))');


% This is the cosine similarity.
costheta = tmp1./tmp2;


% We convert to a distance matrix.
% This has the distances in the full space.
Df = 1 - costheta;
```

```
Df(logical(eye(size(Df)))) = 0;
```

The matrix **`Df`** is an n x n matrix of pairwise cosine distances. The diagonal elements of this matrix have to be zero. Some might not be exactly zero due to numerical errors. Thus, we first set those diagonal elements to zero.

Random Projection: Example

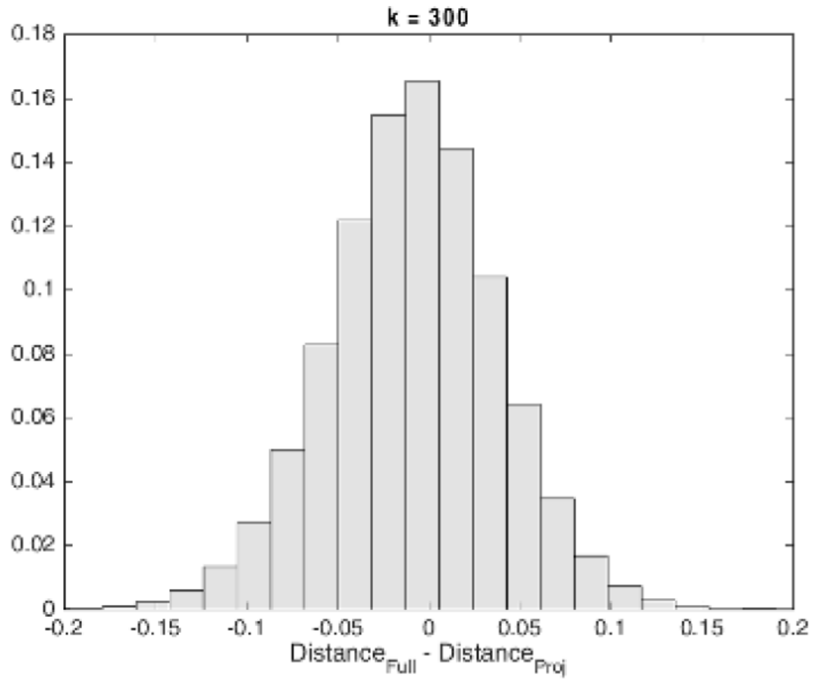
```
% We choose the dimensionality we are projecting to.
d = 300;
% Create the random projection matrix.
R = sparse(randn(p,d));
% We produce a matrix which is 7505 by 300.
testspproj = ((1/sqrt(d))*R'*testsp')';
```

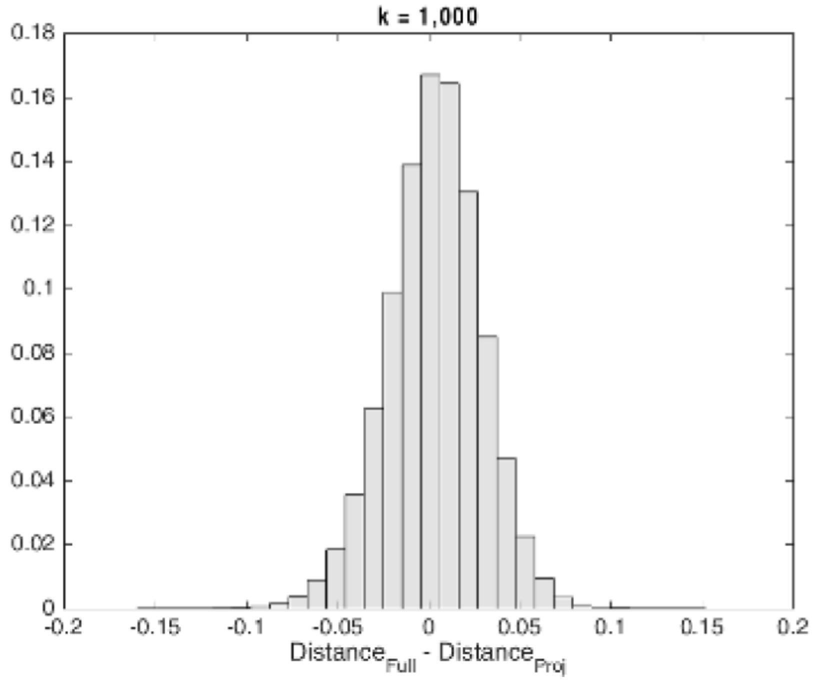
We now calculate the pairwise cosine distances in the subspace.

```
% Calculate the cosine distance for the projected data.
tmp1proj = testspproj * testspproj';
tmp2proj = ...
    sqrt(diag(tmp1proj))*sqrt((diag(tmp1proj))');
costhetaproj = tmp1proj./tmp2proj;
Dproj = 1 - costhetaproj;
```

```
% Compare using the difference in the distances.
D = Df - Dproj;
```

Random Projection: Example

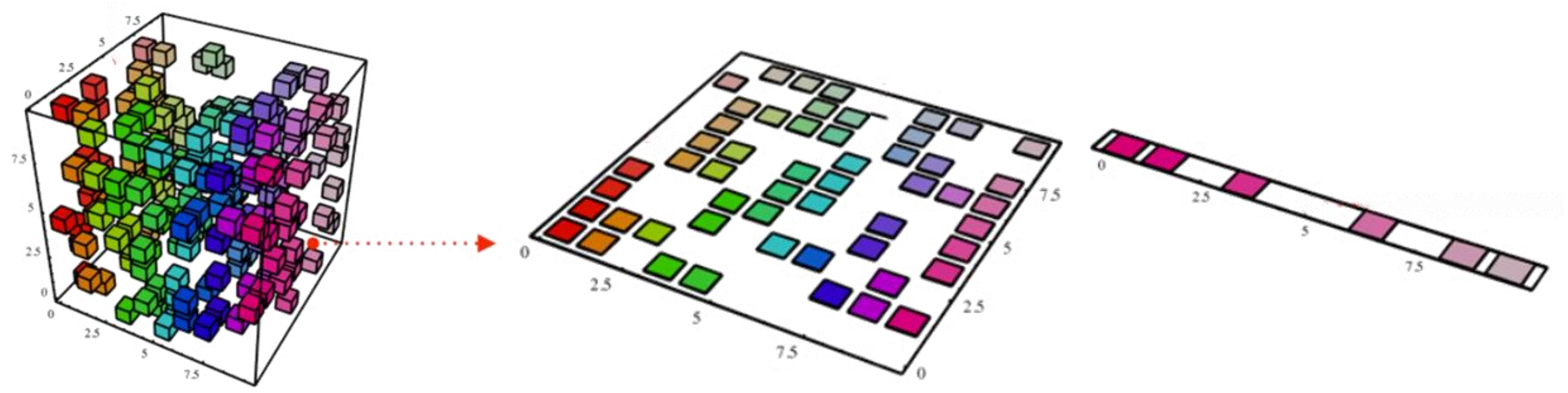




These plots show the distribution of the distances before and after projecting to a lower dimensional space. The first one projects to 300 dimensions and the second to 1,000. Both show that the average difference is close to zero. We also see that the distances tend to be closer (less variance) when we use more dimensions.

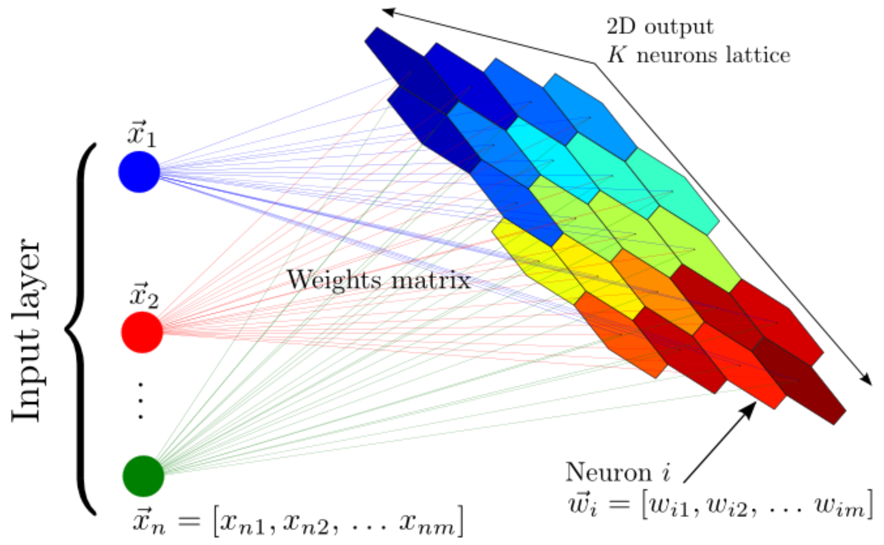
Self Organizing Maps (SOM)

- Neural Networks use processing, inspired by the human brain, as a basis to develop algorithms that can be used to model and understand complex patterns and prediction problems.
- There are several types of neural networks and each has its own unique use.
- The Self Organizing Map (SOM) is one such variant of the neural network, also known as Kohonen's Map.
- It is an unsupervised neural network that is trained using unsupervised learning techniques to produce a low dimensional, discretized representation from the input space of the training samples, known as a map and is, therefore, a method to reduce data dimensions.



Self Organizing Maps (SOM)

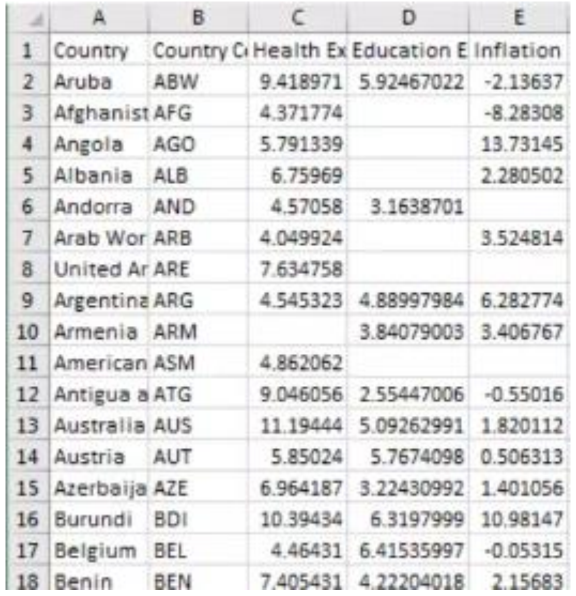
- Provide a way of representing multidimensional data in lower dimensional spaces, usually **two** dimensions
- "Self Organizing" because there is no ***supervision***
- Neurons that lie close to each other represent clusters with similar properties


2D output
K neurons lattice
$\vec{x}_1$
$\vec{x}_2$
Input layer
Weights matrix
Neuron i
$\vec{w}_i = [w_{i1}, w_{i2}, \dots w_{im}]$
$\vec{x}_n = [x_{n1}, x_{n2}, \dots x_{nm}]$

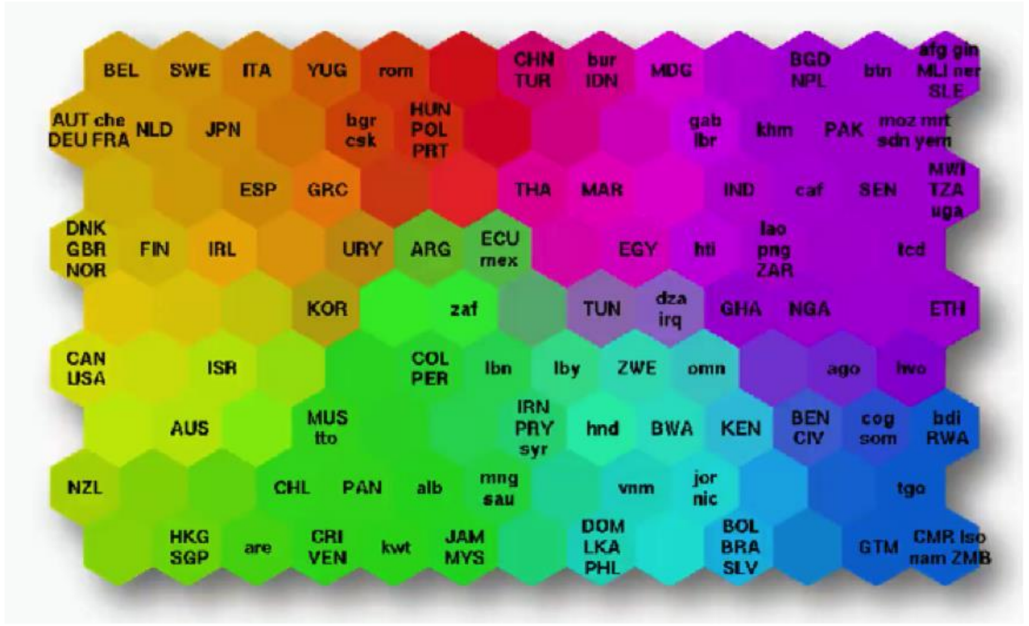
The example shows a complex data set consisting of a massive amount of columns and demonstrates how that data set's dimensionality can be reduced.

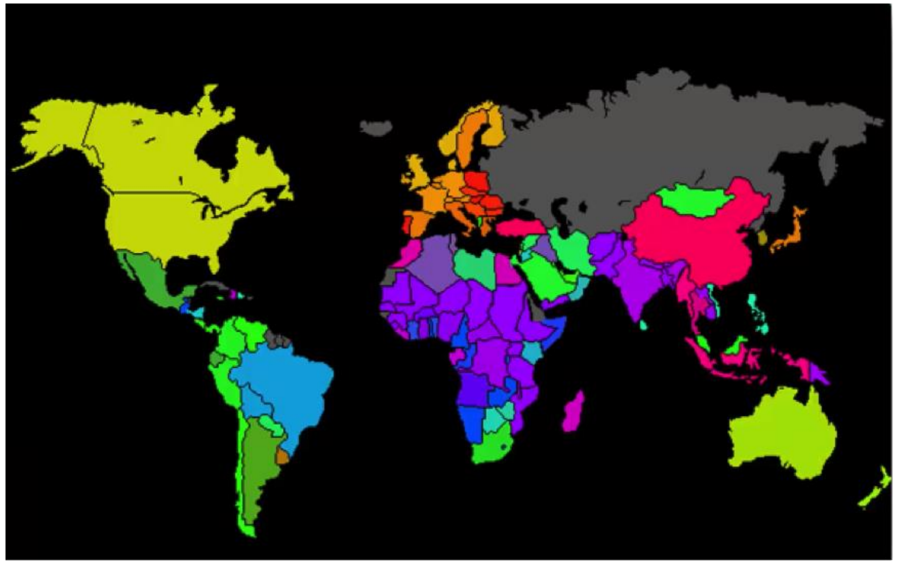
So, instead of having to deal with hundreds of rows and columns, the data is processed into a simplified 2D map; that's what we call a self-organizing map. The map provides you with a two-dimensional representation of the exact same data set; one that is easier to read.

Self Organizing Maps (SOM)



	A	B	C	D	E
1	Country	Country C	Health Ex	Education E	Inflation
2	Aruba	ABW	9.418971	5.92467022	-2.13637
3	Afghanist	AFG	4.371774		-8.28308
4	Angola	AGO	5.791339		13.73145
5	Albania	ALB	6.75969		2.280502
6	Andorra	AND	4.57058	3.1638701	
7	Arab Wor	ARB	4.049924		3.524814
8	United Ar	ARE	7.634758		
9	Argentina	ARG	4.545323	4.88997984	6.282774
10	Armenia	ARM		3.84079003	3.406767
11	American	ASM	4.862062		
12	Antigua a	ATG	9.046056	2.55447006	-0.55016
13	Australia	AUS	11.19444	5.09262991	1.820112
14	Austria	AUT	5.85024	5.7674098	0.506313
15	Azerbaija	AZE	6.964187	3.22430992	1.401056
16	Burundi	BDI	10.39434	6.3197999	10.98147
17	Belgium	BEL	4.46431	6.41535997	-0.05315
18	Benin	BEN	7.405431	4.22204018	2.15683





- In this case, we got the data set from the World Bank containing all 39 indicators of human development, processed them into a SOM, and then applied that to a world map

Self Organizing Maps (SOM)

- There are three main ways in which a Self-Organizing Map is different from a “standard” ANN:
  - A SOM is not a series of layers, but typically a 2D grid of neurons
  - They don’t learn by error-correcting, they implement something called competitive learning
  - They deal with unsupervised machine learning problems

- Competitive learning in the case of a SOM refers to the fact that when an input is “presented” to the network, only one of the neurons in the grid will be activated. In a way the neurons on the grid “compete” for each input.

- The unsupervised aspect of a SOM refers to the idea that you present your inputs to it without associating them with an output. Instead, a **SOM is used to find structure in your data**.

SOM: Steps

1. Initialize each node’s weight vector, with a random real number

2. Take a random input vector from the training data set and feed into input layer

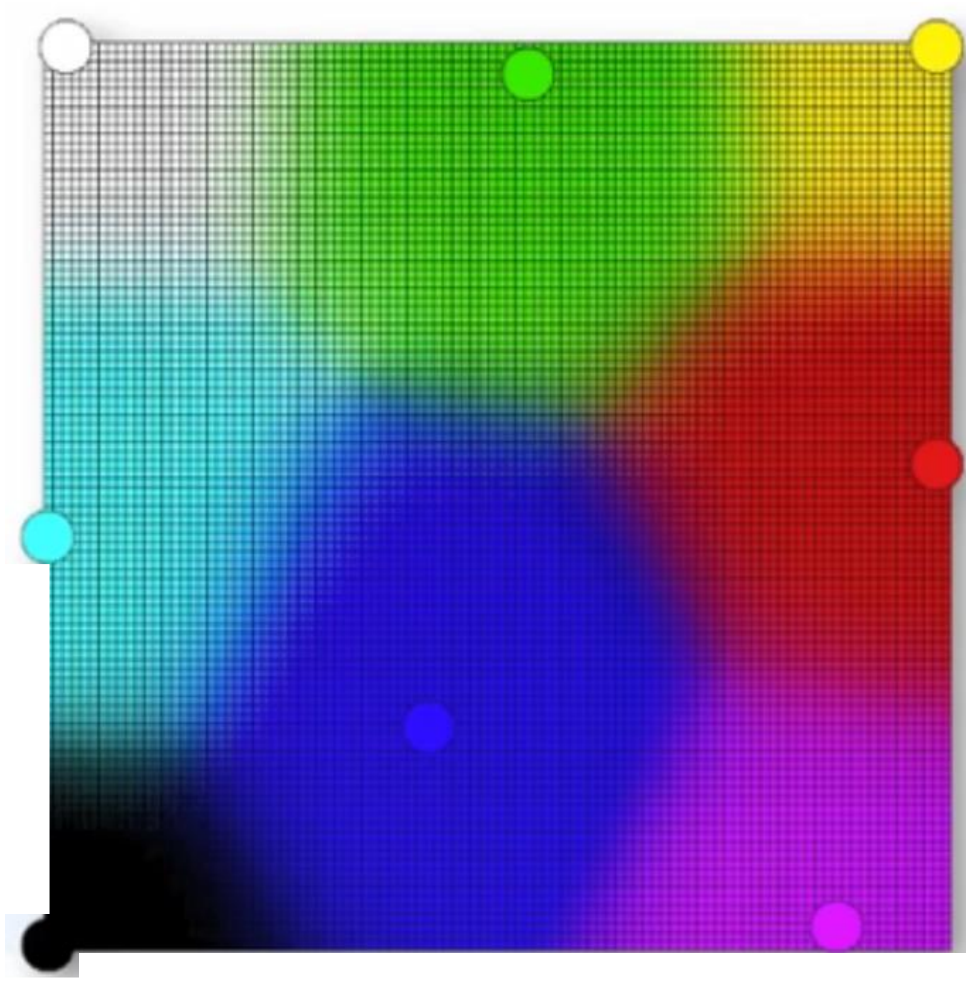
3. Every node in the network is examined to calculate which ones’ weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU)

4. Determine the neighborhood(radius) of BMU, diminishing each iteration

5. Reward the nodes by adjusting weight vector, of each node within the neighborhood determined in step 4

6. Return to step 2 and repeat the procedure n times

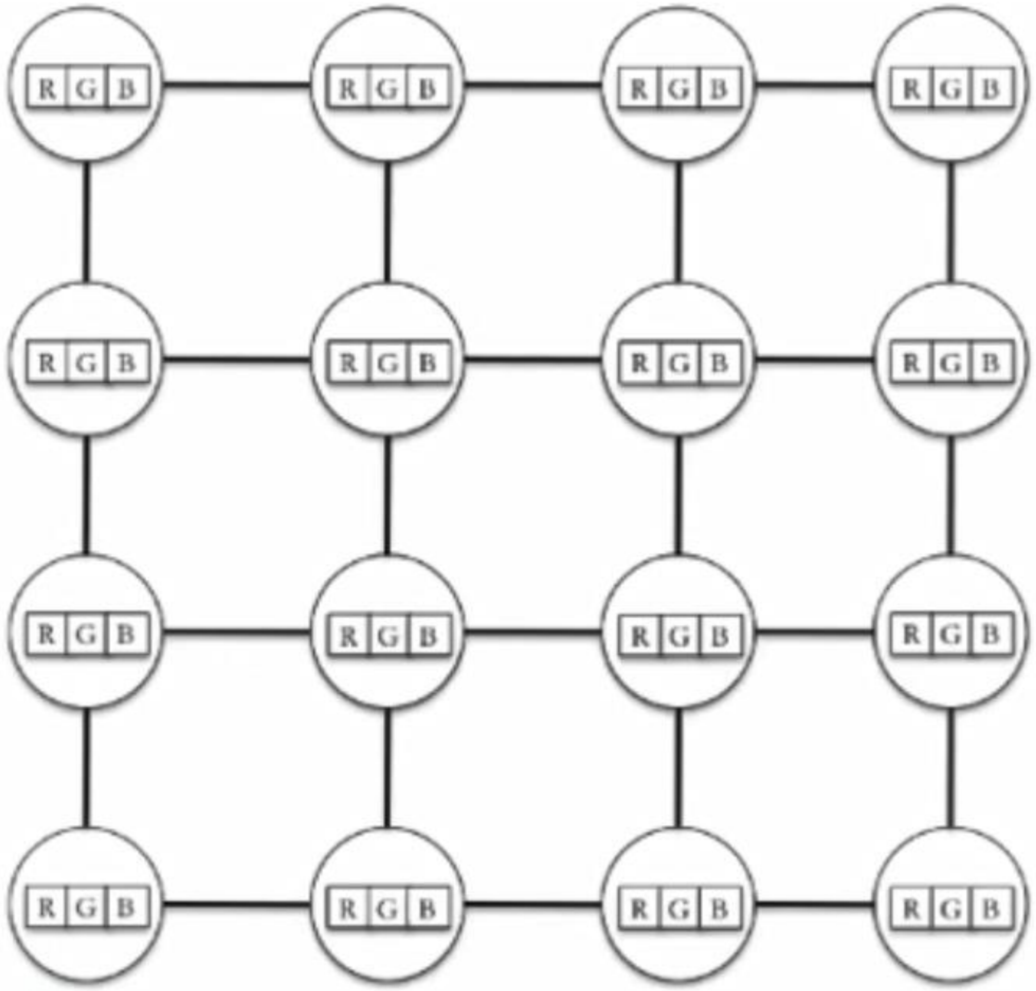
SOM: Example



$$x = \begin{array}{cccc} R & G & B & \\ \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}\right] & & & \begin{matrix} ● \\ ● \\ ● \\ ● \\ ● \\ ● \\ ● \\ ○ \end{matrix} \end{array}$$

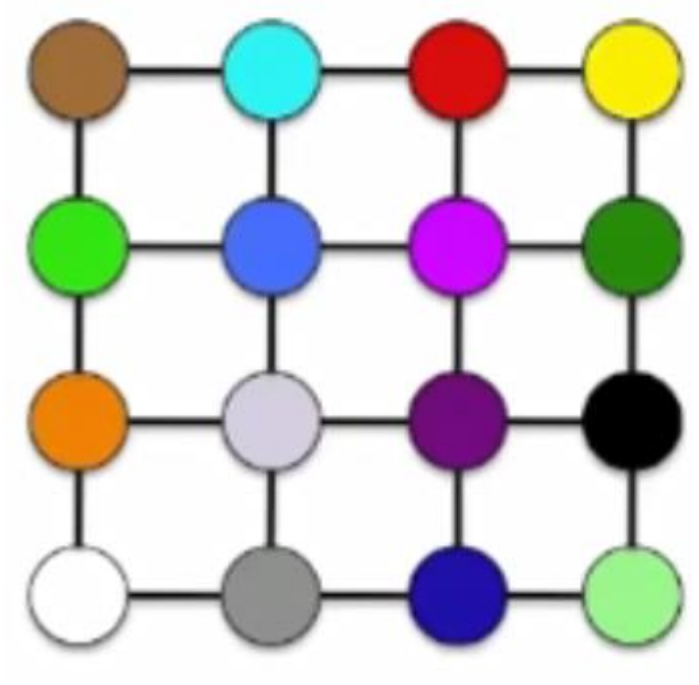
SOM: Example

Grid



SOM: Example

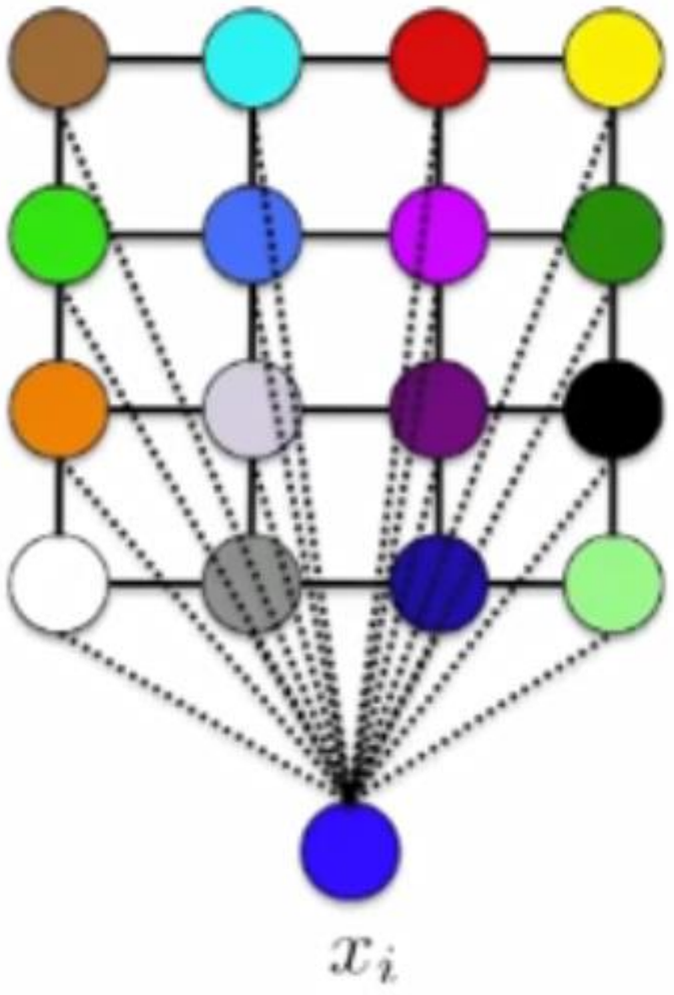
Initialize each node with random features (colors).



SOM: Example

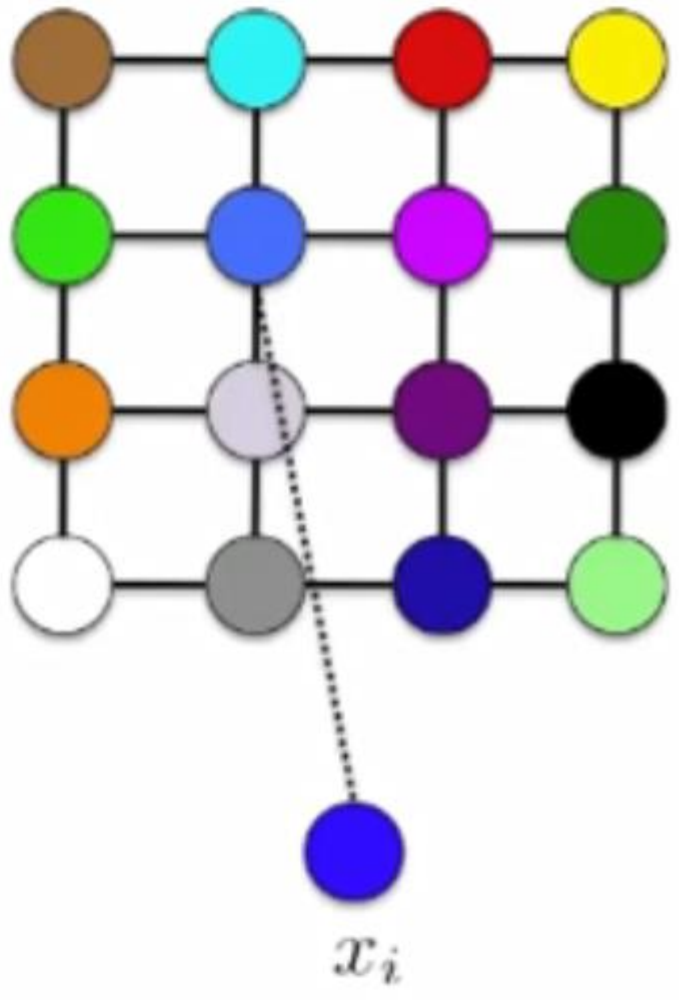
Compare each input with each node in the grid by estimating the Euclidean distance:

$$D_{ij} = \sqrt{\sum_{k=1}^{m} (x_{ik} - N_{jk})^2}$$



SOM: Example

Select the node with minimum distance or the best matching unit (BMU).



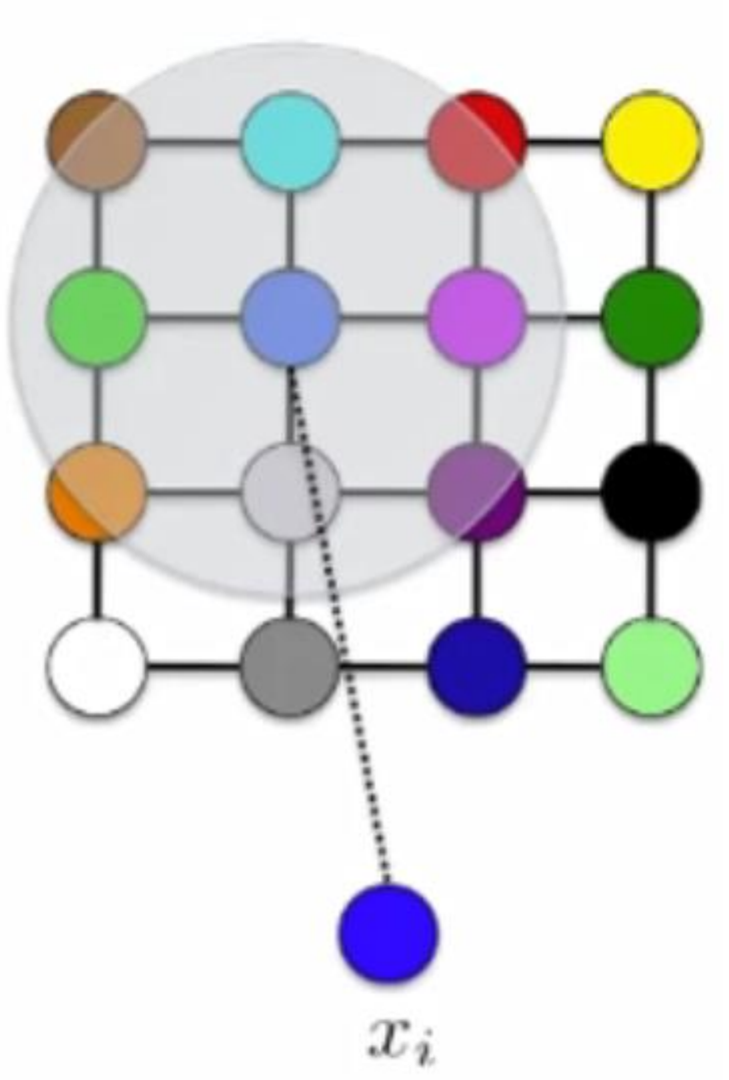
SOM: Example

Update

- For each node in the range of BMU

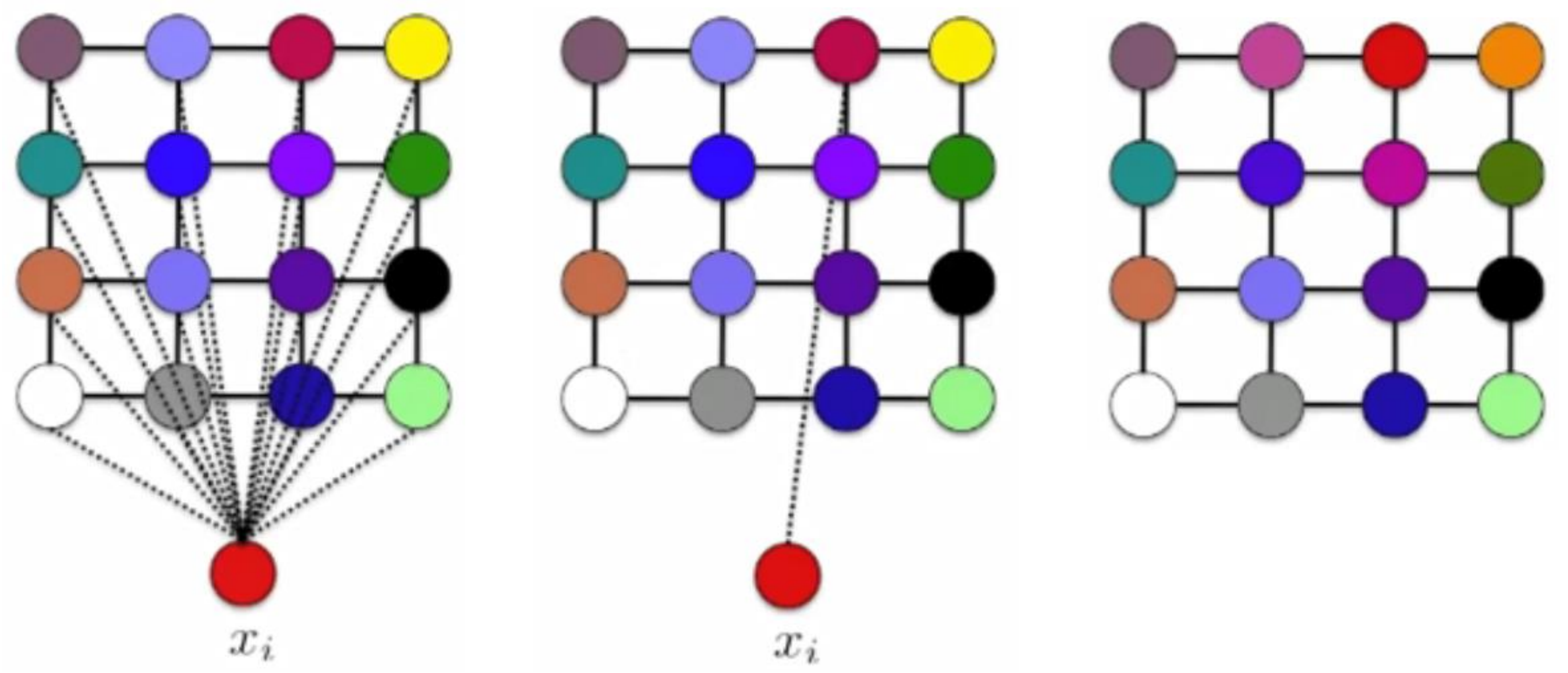
$$N_j = N_j + \eta w_j \left(x_i - N_j\right)$$

- Where w_j is a weight parameter that depends on the distance between the node and the BMU, and η is the learning rate.



SOM: Example

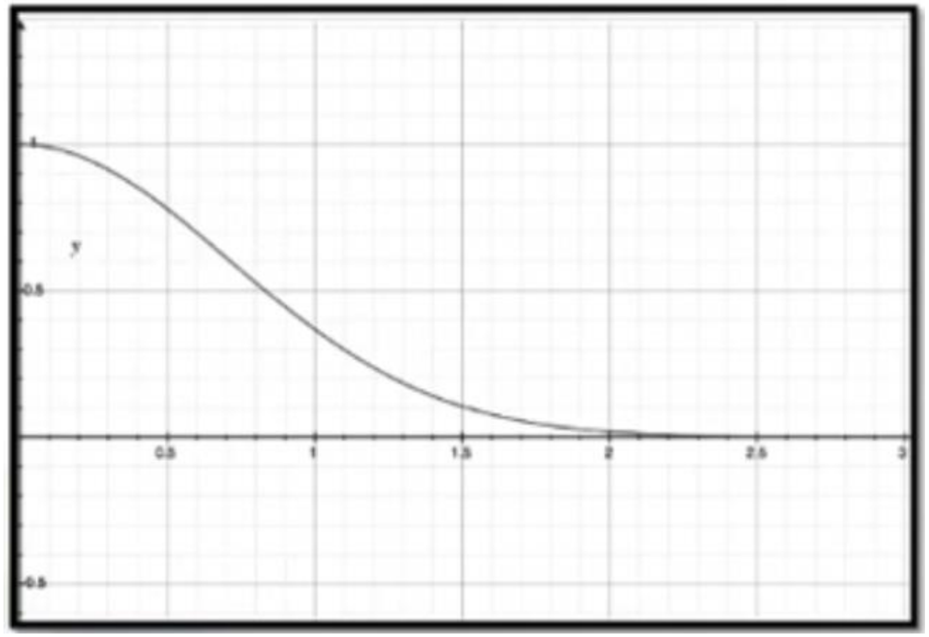
Repeat the process with different input.

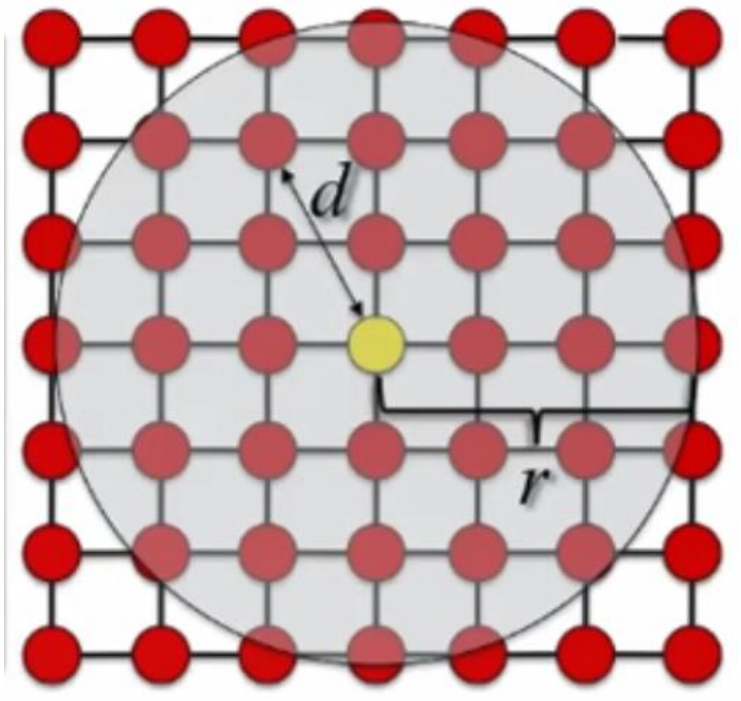


SOM: Example

Weight Decay

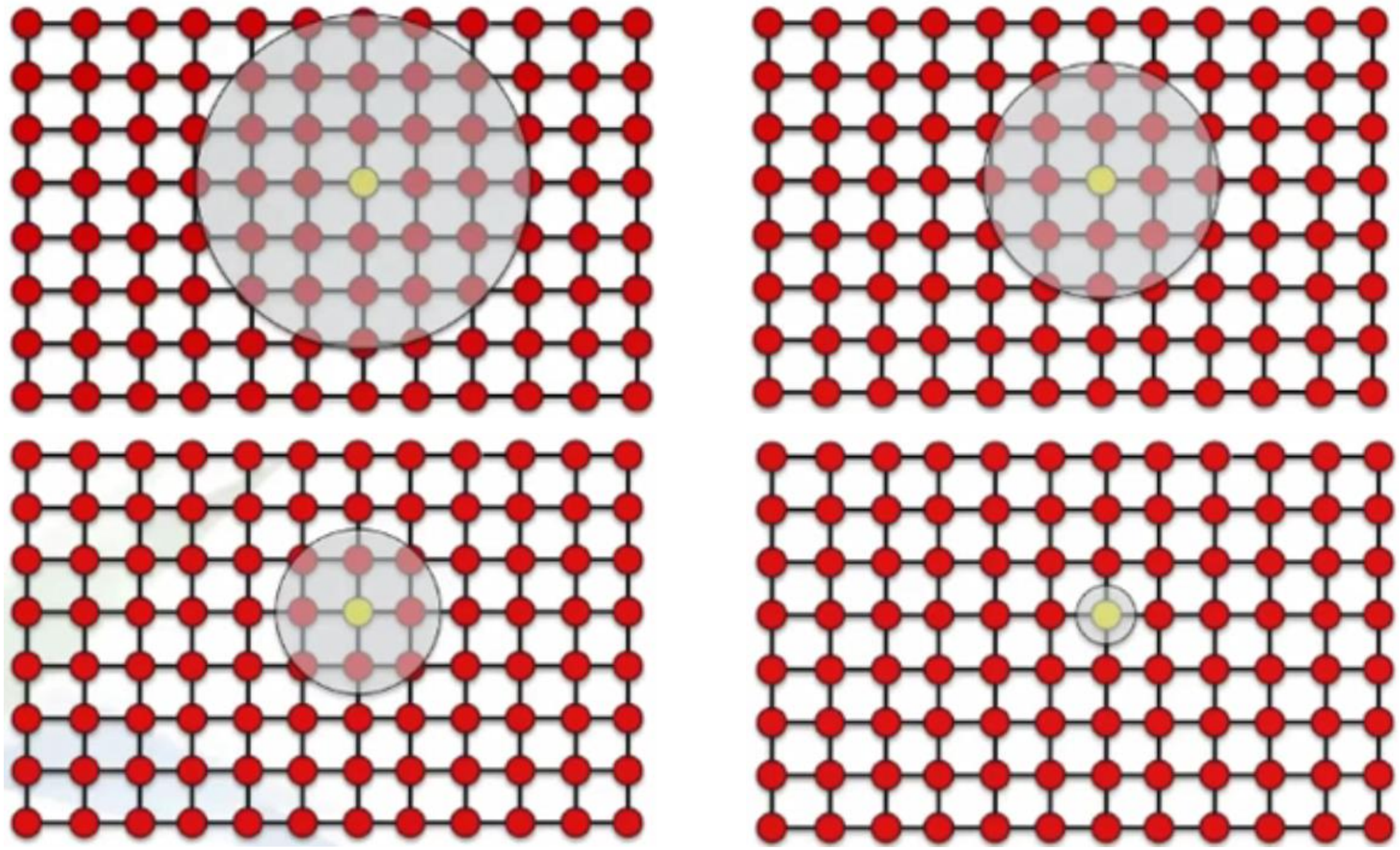
$$w_j = e^{-\frac{d_j^2}{2r^2}}$$





SOM: Example

Repeat with Smaller Radius



SOM: Example

How to determine r

$$max = 100$$

$$r_t = (\text{Grid Size})\, f(t)$$

$$f(0) = 1 \qquad f(max) = \frac{1}{\text{Grid Size}}$$

$$f(t) = e^{-t\frac{ln(\text{Grid Size})}{max}}$$

Where max refers to the maximum number of iterations.

Thank You

&

Questions