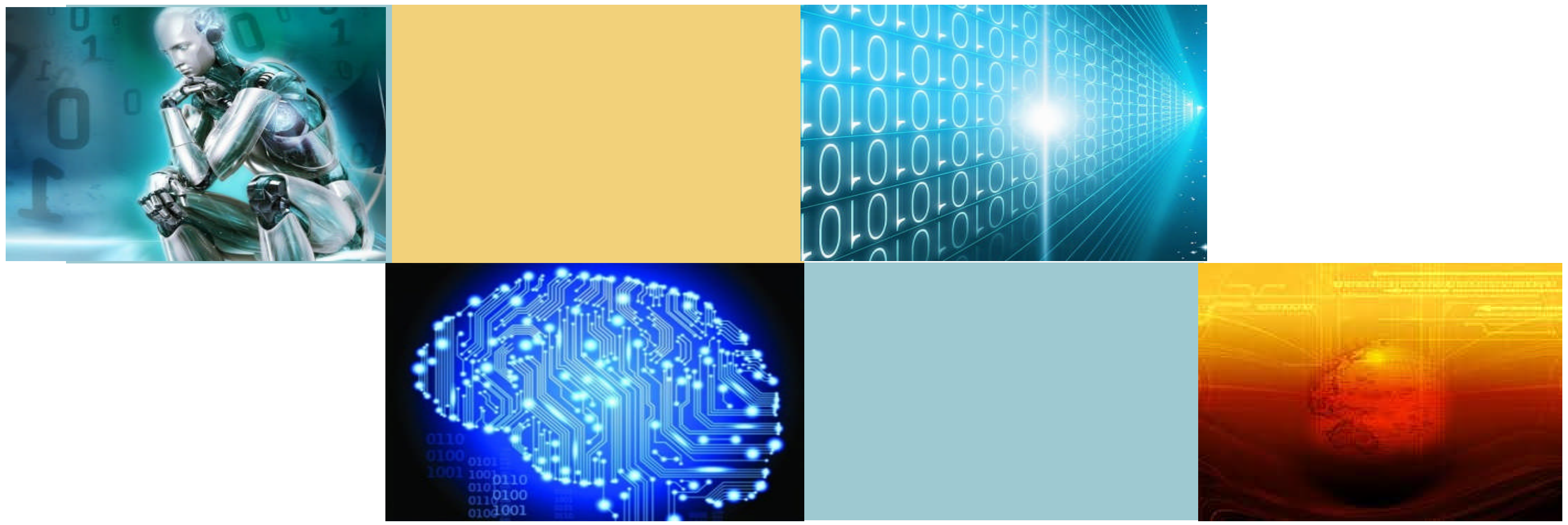
Diffusion Algorithms –
Crucial for Efficient Solutions to Machine Learning Problems



Goutam Chakraborty

Iwate Prefectural University, Japan

MITS, India

DBKDA 2025, March 2025, Lisbon

Mathematics & Physics the precursors of AI

- Early AI 1950 to 1979 – mostly logic and statistics:
  - 1950: Alan Turing wrote "Computer Machinery and Intelligence"
  - 1958: John McCarthy created LISP, a functional programming language
  - 1959 to 1979 - Expert Systems, simple robots. AAAI founded in 1979
- 1980 to 1993: Cold and hot waves of AI
  - Minsky Perceptrons Book – Cold wave,
    Hopfield & Tank - Optimization, Associative Memory 1985, Rumelhart MLP Back Propagation Training 1987 - for classification as well as (sparse) Autoencoder (BP without a teacher) – Hot Wave

The above tools are based on logic, probability, Linear Algebra, Optimization using Gradient Descent and Lagrangian Functions (Hopfield Net – Solving Optimization Problems and Associative Memory)

Some nice tools to play with

- AI Boom 1994 to 2011 – A transition from Domain specific AI to General-purpose AI and bringing more ideas from physics
  - 1995: Kohonen SOM for visualization of data distribution,
    First GPU used in Sony Play Station I
  - 1997: Deep Blue (rule based parallel search), Speech recognition (HMM),
    RNN-LSTM,
  - 2000: Restricted Boltzmann Machine (RBM) – an autoencoder model,
    CNN for training filters instead of manual design,
  - 200X: **Diffusion Maps** – A global geometric Framework for **Nonlinear Dimensionality Reduction** – Lafon, Coifman, Joshua H et.al. Also used for data visualization.
    Google Page-search (different from content based search),
    Evolution of **Scale Free Social Network applications** – **Network Clustering**
    Facebook, Twitter, Collaborative filtering, Disease spread, ….
  - 2010-11: MS Kinect, ChatBots - IBM Watson, Siri

Super General AI (SGAI) and Threat to Human Society (starting with job erosion)

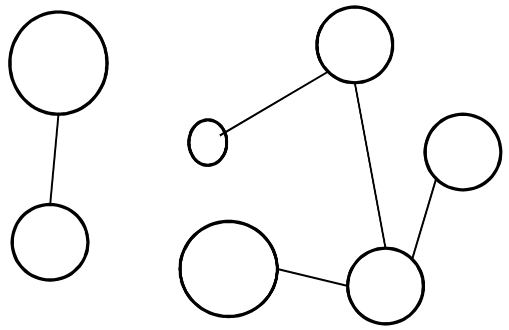
- **Invasion of our brain and society by SGAI (like super-special hospitals!!):**
  - 2013: Variational Autoencoder for data (image) generation
  - 2014: Attention Model to address remembering contexts in LLM. Generative Adversarial Network (GAN) a generator-discriminator network model.
  - 2015: U-Net (U for the shape) for bio-medical image segmentation. It is an auto-encoder and can be used for feature compression.
    Res-Net with skip connection to enrich feature and avoid vanishing gradient problem.
  - 2015: **Diffusion models to** learn complex probability distribution from data using techniques from non-equilibrium thermodynamics of diffusion.
  - 2016: V-Net for Volumetric Medical Image Segmentation (MRI and CAT images)
  - 2017: “Attention is all you need“ Beginning of Transformer Model for image generation
  - 2022~: ChatGPT, Gemini, Jasper, Claude…..

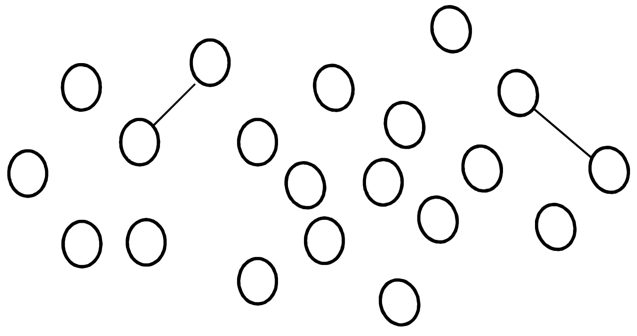
Today's Agenda

- Fundamentals:
  - Diffusion and Laplacian
  - Graph Laplacian
  - Laplacian of a Function
- Graph Laplacian:
  - Network (graph) Clustering
  - Matrix Rank Reduction and Matrix Completion Problems
  - Miscellaneous applications of Diffusion on Networks
- Human and Computer vision:
  - Center surround, Edge Detection and Laplacian Filter
- Diffusion and Image Enhancement
- Diffusion on a Graph
- Diffusion and Image Generation

Diffusion, Percolation and the Network (Graph)

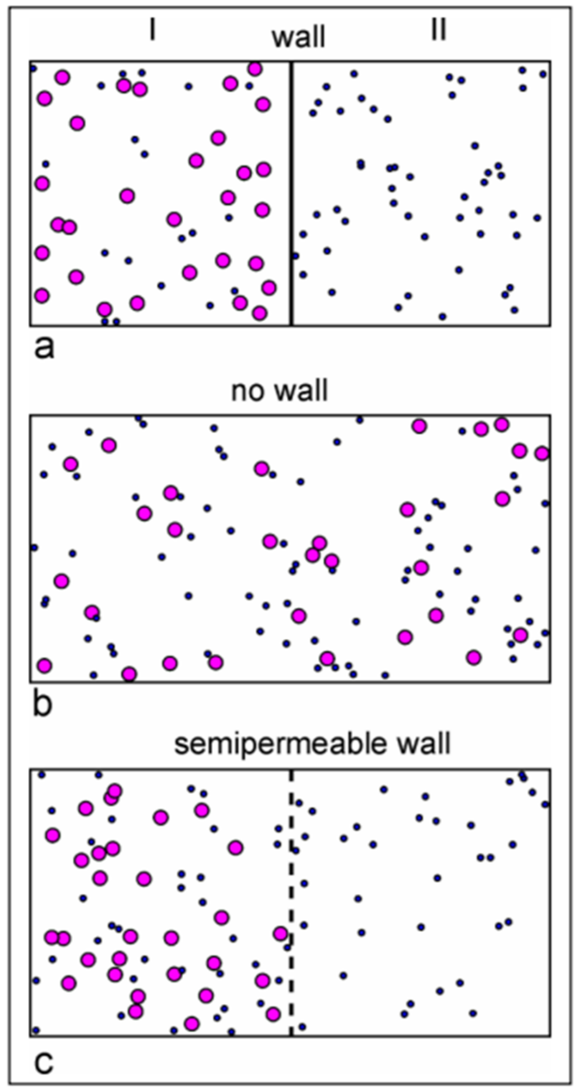
There are many physical phenomena in which a fluid spreads randomly through a medium. Consider the fluid as data, or virus. This spread depends on the nature of the (1) fluid (say viscosity) and to the (2) medium (say porosity).
(1) is known as diffusion, and (2) as percolation.
Examples are how a virus or a news spreads in a population.





- The mediums (percolation part), like permeability of rocks, is represented by graphs with different node capacities and edges (connection weights).
- The diffusion part depends on the fluid viscosity, the mechanism a virus or a fake Instagram posting spreads.
- Interesting things happen when the network is scale-free. All naturally evolved networks are scale-free.

Fundamentals: Diffusion and Laplacian

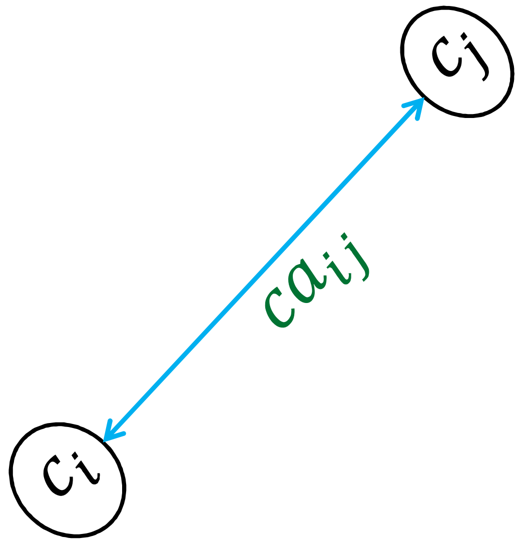


Diffusion of Particles

The temporal and spatial change of Concentration can be determined using diffusion Equation

$$\frac{\partial c}{\partial t} = \rho\, \nabla^2 c = \rho \times div \cdot grad(c) = \rho \times Laplacian(c)$$

For 1-dimension $\frac{\partial c}{\partial t} = \rho \frac{\partial^2 c}{\partial x^2}$ and for 3-dimension $\frac{\partial c}{\partial t} = \rho \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} + \frac{\partial^2 c}{\partial z^2} \right)$

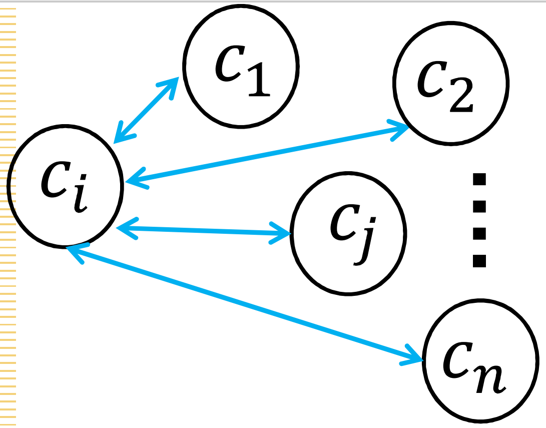


Diffusion between two nodes *i* and *j*

$$\frac{\partial c_i}{\partial t} = \rho\, a_{ij}(c_j - c_i) \text{ and } \frac{\partial c_j}{\partial t} = \rho a_{ji}(c_i - c_j)$$

where $a_{ij} = a_{ji} = 1 \; or \; 0,$ depending on whether the nodes are connected or not. *c* is the rate of flow.

Fundamentals: Diffusion on a Graph



Diffusion to i^{th} node is from all other nodes

$$\frac{\partial c_i}{\partial t} = \rho \sum_{j=1}^{n} a_{ij}(c_j - c_i) = \rho \sum_{j=1}^{n} a_{ij} c_j - \rho \sum_{j=1}^{n} a_{ij} c_i$$

where $a_{ij} = a_{ji} = 1 \; or \; 0$, ρ is the rate of flow.

$$\frac{\partial c_i}{\partial t} = \rho \sum_{j=1}^{n} a_{ij} c_j - \rho c_i \times d_i = \rho \sum_{j=1}^{n} a_{ij} c_j - \rho \sum_{j=1}^{n} \delta_{ij} c_j d_j; \; \delta_{ij} \; delta \; function$$

$$\frac{\partial c_i}{\partial t} = \rho \sum_{j=1}^{n} a_{ij} c_j - \rho \sum_{j=1}^{n} \delta_{ij} c_j d_j = \rho[A_i \vec{c}] - \rho[D_i \vec{c}]$$

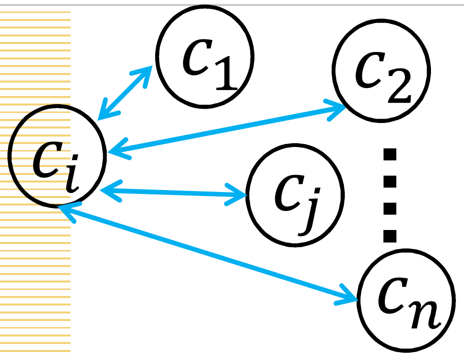
$$\vec{c} = \begin{bmatrix} c_1 \\ \cdots \\ c_n \end{bmatrix};$$

A_i is i-th row of the adjacency matrix, D_i is the i-th row of Degree matrix

$$\frac{\partial \vec{c}}{\partial t} = \rho[A\vec{c}] - \rho[D\vec{c}] = \rho[A - D]\vec{c}$$

$$\frac{\partial \vec{c}}{\partial t} = \rho[A\vec{c}] - \rho[D\vec{c}] = \rho[A - D]\vec{c}$$

Fundamentals: Diffusion on a Graph



$$\frac{\partial \vec{c}}{\partial t} = \rho[A\vec{c}] - \rho[D\vec{c}] = \rho[A - D]\vec{c}$$

$$\frac{\partial \vec{c}(t)}{\partial t} = \alpha c;\ \vec{c}(t) = \vec{c}(0)e^{-\alpha t}$$

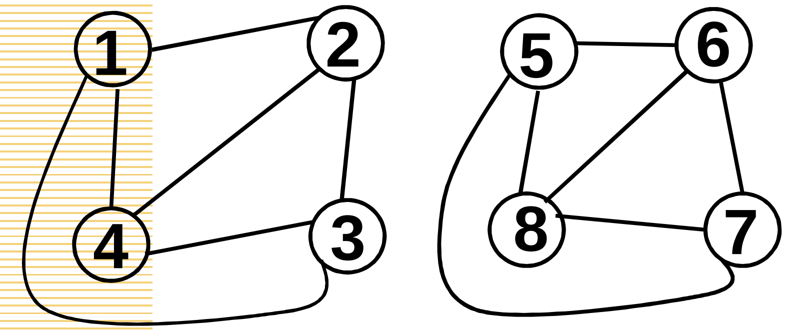
$$\frac{\partial \vec{c}}{\partial t} + \rho[D - A]\vec{c} = 0;\ \frac{\partial \vec{c}}{\partial t} + \rho[L]\vec{c} = 0;$$

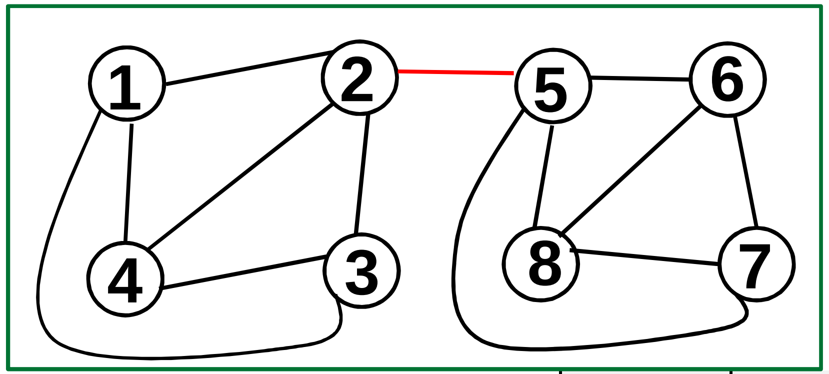
$$\mathit{When\ the\ system\ converges}\ \frac{\partial \overrightarrow{c_\infty}}{\partial t} = 0;\ \rho[L]\overrightarrow{c_\infty} = [L]\overrightarrow{c_\infty} = 0;$$

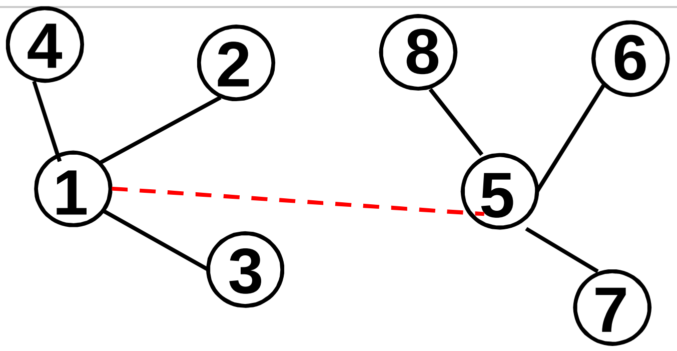
A and D are symmetric. [D - A] is symmetric. All elements of L are REAL.
$[D - A]$ ***is positive definite or positive semidefite***;
All Eigenvalues are non − negative reals. Eigenvectors orthogonal.

As every row of [D - A] sums up to 0, there is a 0 eigenvalue, and corresponding eigenvector is a vector with all elements equal.
$\overrightarrow{c_\infty}$ **is that eigenvector and corresponding eigenvalue = 0;**
The other eigenvectors are interesting, they partitions the network.
The method is known as **FIELDER's "Spectral Graph Partitioning."**
Spectral Graph Partitioning is different from Matrix Spectral Analysis.

Let's do an exercise on Fielder "Graph Spectrum Analysis" (Not to confuse with Matrix Spectral Analysis)







Laplacian Matrix L

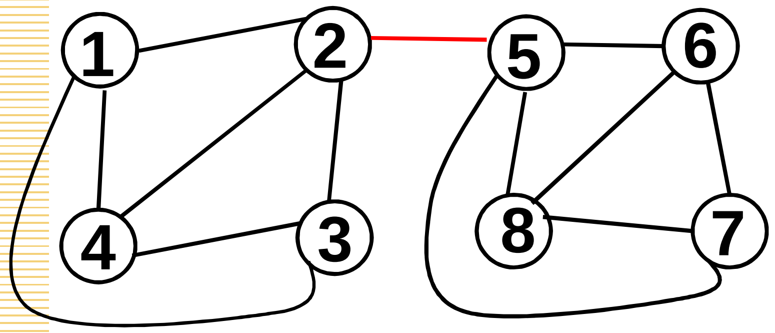
3	-1	-1	-1	0	0	0	0
-1	4	-1	-1	-1	0	0	0
-1	-1	3	-1	0	0	0	0
-1	-1	-1	3	0	0	0	0
0	-1	0	0	4	-1	-1	-1
0	0	0	0	-1	3	-1	-1
0	0	0	0	-1	-1	3	-1
0	0	0	0	-1	-1	-1	3

Eigen Values and Eigen Vectors (normalized) of the Laplacian Matrix L

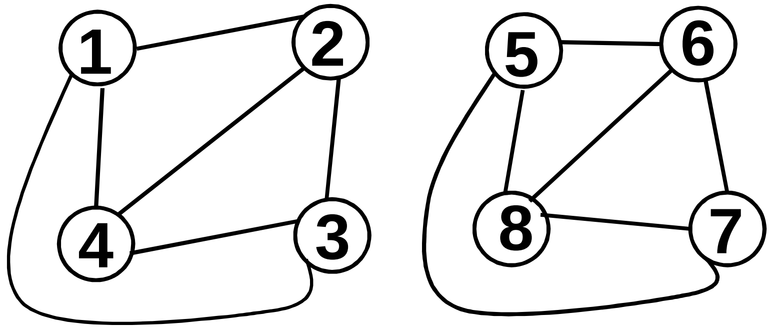
$\lambda = 0$	.35	4.0					
.35	-.38						
.35	-.25						
.35	-.38						
.35	-.38						
.35	.25						
.35	.38						
.35	.38						
.35	.38						

Laplacian Matrices are Sparse. Very efficient Algorithms to find Eigen Pairs

Graph Spectrum Analysis



When all nodes are connected, only one partition: λ_1 is 0, $\lambda_2, \lambda_3 \ldots . > 0$; Sign of the elements of $\overrightarrow{v_2}$ will divide the graph into two partitions.



$$L = \begin{vmatrix} \begin{vmatrix} L_1 \end{vmatrix} & \mathbf{0} \\ \mathbf{0} & \begin{vmatrix} L_2 \end{vmatrix} \end{vmatrix}$$

When two disconnected partitions: $\lambda_1 = \lambda_2 = 0$; $\lambda_3 \ldots . > 0$; Sign of the elements of $\overrightarrow{v_3}$ will divide the graph into two partitions. Elements of further eigen vectors will give a hierarchical clustering.

Clustering of Term-Document Matrix (Utility Matrix is a different Story)

Term / Document	1	2	3	4	…	N
Customer1	0.05	0.01	0.05	0		0.1
Customer2	0	0	0.001	0		0
Customer3	0.07	0	0.07	0.02		0
Customer4	0	0	0	0.01		0.01
…						
CustomerM	0	0	0	0		0.02

Item / Customer	1	2	3	4	…	N
Customer 1	1	1	1			1
Customer 2			1			
Customer 3	1		1	1		1
Customer 4				1		1
…						
Customer M						1

N terms in M different documents is the term-document Matrix.
N books and the purchase history of M customers.
N movies and rating information from M customers.
We need to cluster the column vectors. Consider the column vectors as nodes, Calculate the adjacency matrix, and Graph Spectral Analysis.

Utility Matrix

- Two types of utility matrix
  - Rating

Movie / Customer	HP1	HP2	HP3	Twilight
Customer1	5	4	5	3
Customer2	5	5		4
Customer3	5	4	5	4
Customer4	3	3	3	5

Assessment and fact: The first matrix is users' Assessments (rating). The empty places are empty. The same in the purchase information, we can't put a zero for Items not purchased. Whether the customer wants to purchase or not is important for A Recommendation System. This is a multi-objective optimization problem – Rank reduction With lowest Forbenius Distance for filled in entries.

- Purchase lists

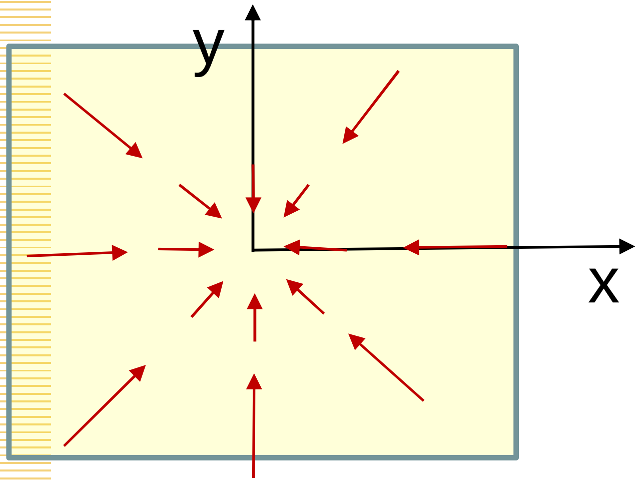
	JAZZ			Indian Classic	
CD / Customer	**CD1**	**CD2**	**CD3**	**CD4**	**CD5**
A	1	1	1		
B	1	1	1		
C	1	1	1		
D	1	1	1		
E			1	1	1
F			1	1	1
G			1	1	1
H			1	1	1

Computer Vision - Edge Detection, Laplacian Filter

- Fundamentals:
  - Diffusion and Laplacian
  - Graph Laplacian
  - Laplacian of a Function
- Graph Laplacian:
  - Network (graph) Clustering
  - Matrix Rank Reduction and Matrix Completion Problems
  - Miscellaneous applications of Diffusion on Networks
- Human and Computer vision:
  - Edge Detection and Laplacian Filter
- Diffusion and Image Enhancement
- Diffusion and Image Generation

Derivative and Laplacian operation – Introduction to preliminary mathematics

- Laplace Transform (LT) is used to convert a function from time domain to frequency domain, much like Fourier transform (FT). FT is a specific case of a LT. Laplacian is an operator.
- Laplacian of a 2-dimension function (x,y) is the divergence $(\nabla \cdot)$ of the gradient of the function,
$$\text{Laplacian } f = \Delta(f) = \nabla \cdot (\nabla f) = \nabla^2 f(x,y)$$
- Function $f(x,y)$ gives a scalar value at every point (x,y). For example, $f(x,y) = 4 - x^2 - y^2$,
$$f(x,y) = 0 \; on \; the \; circle \; x^2 + y^2 = 4 = 2^2;$$
$$f(x,y) = 3 \; on \; the \; circle \; x^2 + y^2 = 1 = 1^2;$$
$$f(x,y) = -5 \; on \; the \; circle \; x^2 + y^2 = 9 = 3^2; \; etc.$$
- $\nabla f(x,y)$ is the slope of the steepest descent of $f(x,y) \; at \; (x,y)$. It has a direction and magnitude, it is a vector. Thus, $\nabla f(x,y)$ creates a vector field on the (x,y) plane.



- $\nabla f(x,y)$ has smaller values near the origin where the function reaches its maximum. As (x,y) is away from the origin, the vector is larger. All vectors in this filed is towards the origin, where the function (x,y) reaches its maximum.
- For this example, the $\nabla f(x,y)$ field vectors are converging to the origin. So, the divergence value of this vector field of the gradient is converging towards the origin, i.e., the divergence value is negative.

Derivative and Laplacian operation – Introduction to preliminary mathematics

- Laplacian of a 2-dimension function (x,y) is the divergence $(\nabla \cdot)$ of the gradient of the function,

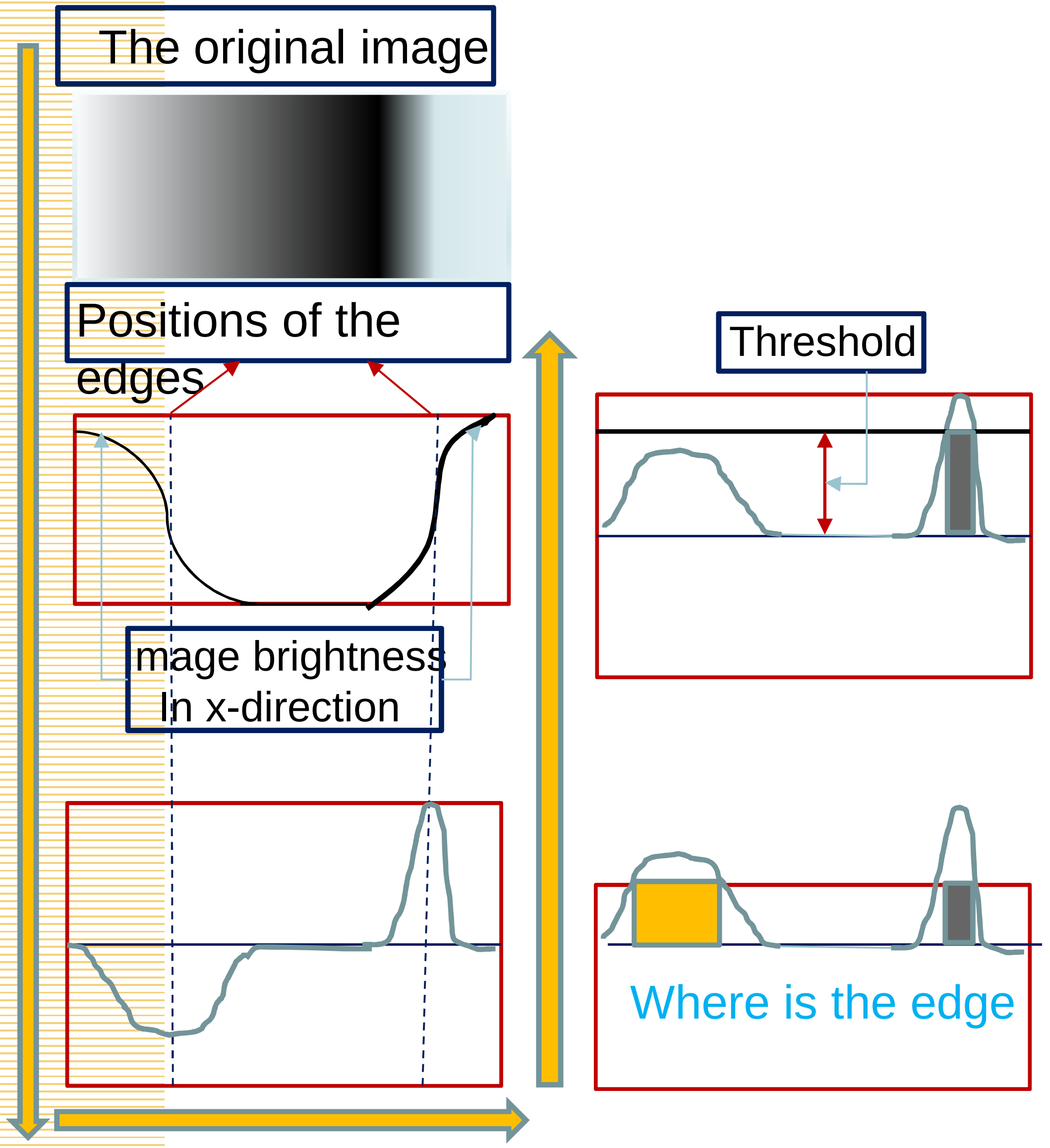
$$\text{Laplacian } f = \Delta(f) = \nabla \cdot (\nabla f) = \nabla^2 f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \frac{\partial^2 f(x,y)}{\partial^2 x} + \frac{\partial^2 f(x,y)}{\partial^2 y} \quad a\ scalar$$

- Exercise: For the Function $f(x,y) = 3 + \sin\frac{x}{2}\cos\frac{y}{2}$, find the maxima and the minima. Find the Laplacian at maxima and minima.
- Laplacian operator can be extended to *n*-dimensions for a function with *n* variables as follows:

$$Laplacian\ f(x_1, x_2, \ldots \ldots x_n) = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}$$

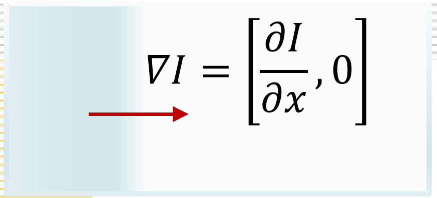
- For an image, it is a 2-dimensional function.

Derivative Kernel for Edge Detection: Edge Detection is the first step in Object Detection
Edges are used to find the contour/boundary of an Object in an image

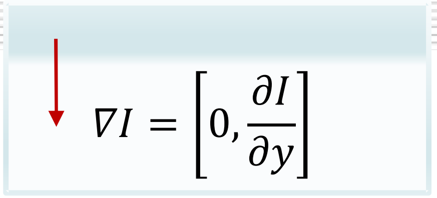


- What is an edge – A rapid change in the image brightness.
- Box-1 the image with two edges, the right edge is sharp.
- Box-2 the Image brightness I as a function $\mathrm{I}(x)$ of the x-coordinate value. It is same at all x values.
- Box-3 is the first derivative of $\mathrm{I}(x)$, ${}^{\partial I}/_{\partial x}$
- $\underset{x}{Argmax} \left(\frac{\partial I}{\partial x}\right)$, the value of x gives the location of an edge
- Box-4 is the plot of $\left|\frac{\partial I}{\partial x}\right|$
- Box-5 is same as Box-4 with a threshold value which crosses $\left|\frac{\partial I}{\partial x}\right|$ at two points. The average gives the location of edge.
- If the peak is lower than the threshold, when the change is slow, we may not consider it as an edge.
- For an image, the dimension is two. We have one

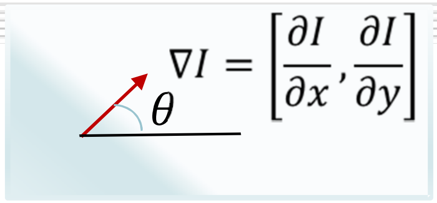
Derivative Kernel (Filter) for Edge Detection:



$\nabla I = \left[\frac{\partial I}{\partial x}, 0\right]$



$\nabla I = \left[0, \frac{\partial I}{\partial y}\right]$



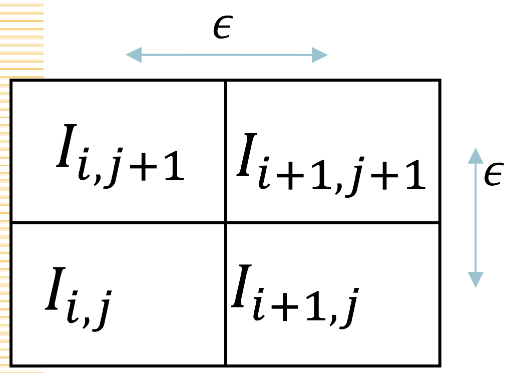
$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$

- Strength of the edge (sharpness) $\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$
- Edge vector orientation (perpendicular to the edge) $\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x}\right)$

Instead of a continuous 2-dim function, an image is a matrix of discrete pixel values

To find derivative, we need at least 2 pixels on the x-axis and y-axis.

ϵ represents the distance between neighboring pixels. In fact, it is just a scale factor.



- $\frac{\partial I}{\partial x} \cong \frac{1}{2\epsilon}\left[\left(I_{i+1,j+1} - I_{i,j+1}\right) + \left(I_{i+1,j} - I_{i,j}\right)\right]$
- $\frac{\partial I}{\partial y} \cong \frac{1}{2\epsilon}\left[\left(I_{i+1,j+1} - I_{i+1,j}\right) + \left(I_{i,j+1} - I_{i,j}\right)\right]$
- Using kernel: $\frac{\partial I}{\partial x} \cong \frac{1}{2\epsilon}$

-1	+1
-1	+1

and $\frac{\partial I}{\partial y} \cong \frac{1}{2\epsilon}$

+1	+1
-1	-1

- Please note that towards (0,0) of the image it is -1

Derivative (Gradient) Kernel for Edge Detection – A few popular kernels:

Gradient

$\frac{\partial I}{\partial x}$

Robert's

0	1
-1	0

Prewitt

-1	0	1
-1	0	1
-1	0	1

Sobel 3×3

-1	0	1
-2	0	2
-1	0	1

Sobel 5×5

-1	-2	0	2	1
-2	-3	0	3	2
-3	-5	0	3	5
-2	-3	0	3	2
-1	-2	0	2	1

In Sobel the central Position is given more importance

$\frac{\partial I}{\partial y}$

Robert's

1	0
0	-1

Prewitt

1	1	1
0	0	0
-1	-1	-1

Sobel 3×3

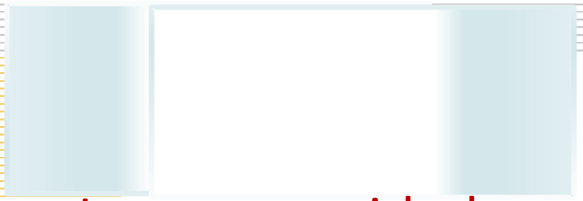
1	2	1
0	0	0
-1	-2	-1

Sobel 5×5

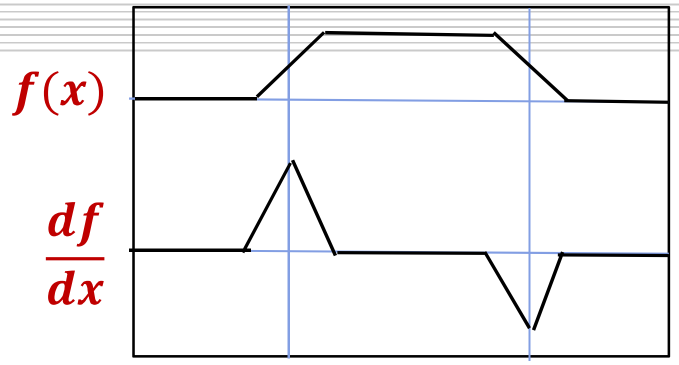
1	2	3	2	1
2	3	5	3	2
0	0	0	0	0
-2	-3	-5	-3	-2
-1	-2	-3	-2	-1

- Defining a threshold T to decide whether there is an edge or not:
- $\|\nabla I(x,y)\| < T \rightarrow Not\ an\ Edge; \|\nabla I(x,y)\| > T \rightarrow Is\ an\ edge$ OR
- $\|\nabla I(x,y)\| < T_1 \rightarrow Not\ an\ Edge; \|\nabla I(x,y)\| > T_2 \rightarrow Is\ an\ edge; T_2 > T_1$
  $If T_1 < \|\nabla I(x,y)\| < T_2, pixel\ is\ an\ edge\ if\ its\ neighbor\ is\ an\ edge.$
- Edge pixels form boundary, and boundary detection is the next important task.

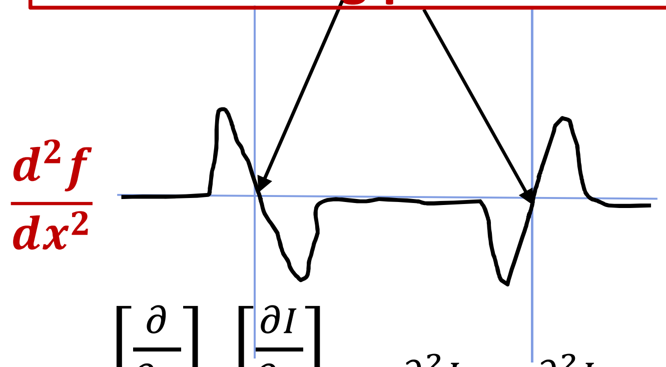
Edge Detection Using Second Derivative: Laplacian Operator



Let us see with the example of an 1-dim signal $f(x)$



Zero-crossing points are edges



For a 2-dimensional Image of continuous signal $I(x,y)$, Laplacian $\nabla^2 I = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} . \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$, is a scalar

Edges are zero-crossing of the Laplacian $\nabla^2 I$. Laplacian does not provide direction of the edge.

ϵ

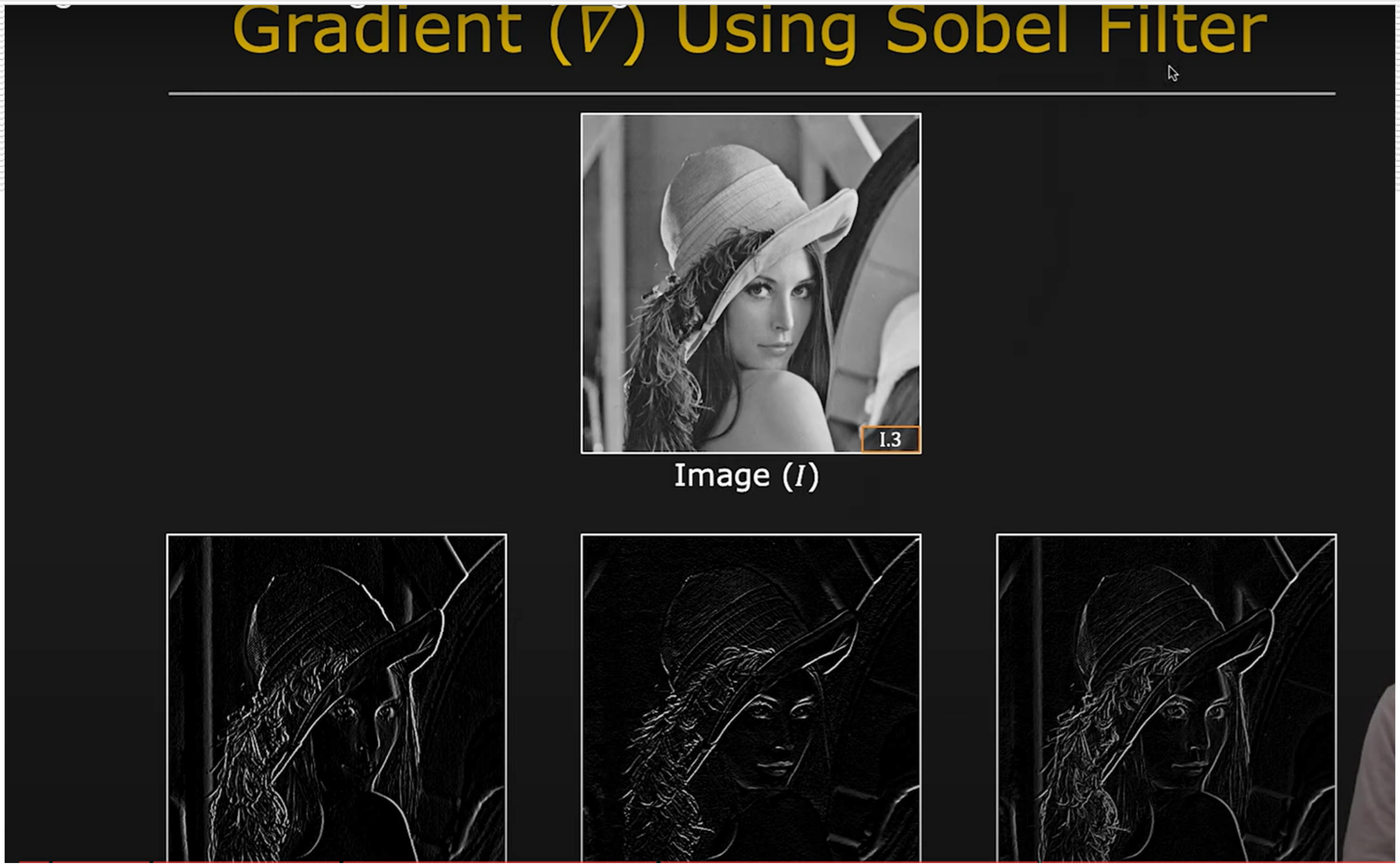
$I_{i-1,j+1}$	$I_{i,j+1}$	$I_{i+1,j+1}$
$I_{i-1,j}$	$I_{i,j}$	$I_{i+1,j}$
$I_{i-1,j-1}$	$I_{i,j-1}$	$I_{i+1,j-1}$

ϵ

$I_{Column\ index,\ Row\ index}$

- $\frac{\partial I^2}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial x}\right) = \frac{1}{\epsilon}\left[\frac{(I_{i+1,j} - I_{i,j})}{\epsilon} - \frac{(I_{i,j} - I_{i-1,j})}{\epsilon}\right] = \frac{1}{\epsilon^2}\ (I_{i-1,j} - 2I_{i,j} + I_{i+1,j})$
- $Similarly,\ \frac{\partial I^2}{\partial y^2} = \frac{1}{\epsilon^2}\ (I_{i,j-1} - 2I_{i,j} + I_{i,j+1})$
- The Kernel $\nabla^2 I = \frac{\partial I^2}{\partial x^2} + \frac{\partial I^2}{\partial y^2} = \frac{1}{\epsilon^2}\ (I_{i-1,j} - 4I_{i,j} + I_{i+1,j} + I_{i,j-1} + I_{i,j+1})$

Sobel filter – gradient edge detector



Gradient (∇) Using Sobel Filter

Image (I)

$\frac{\partial I}{\partial x}$ $\frac{\partial I}{\partial y}$ *Gradient Magnitute*

Edge Detection Using Second Derivative: Laplacian Operator and its kernels

$$\frac{\partial I^2}{\partial x^2} = \frac{1}{\epsilon^2}\left(I_{i-1,j} - 2I_{i,j} + I_{i+1,j}\right)$$

$$\frac{\partial I^2}{\partial x^2} = \frac{1}{\epsilon^2}\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 1 & -2 & 1 \\ \hline 0 & 0 & 0 \\ \hline\end{array}$$

Double derivative in the horizontal direction

$$\frac{\partial I^2}{\partial y^2} = \frac{1}{\epsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right)$$

$$\frac{\partial I^2}{\partial y^2} = \frac{1}{\epsilon^2}\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline\end{array}$$

Double derivative in the vertical direction

$$\textit{The Laplacian Kernel } \nabla^2 I = \frac{\partial I^2}{\partial y^2} + \frac{\partial I^2}{\partial y^2}$$

$$\nabla^2 I = \frac{1}{\epsilon^2}\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array} \quad \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline\end{array}$$

Both are valid Laplacian Kernels

Other Laplacian Kernels are – This is CENTER-SURROUND

1	1	1
1	-8	1
1	1	1

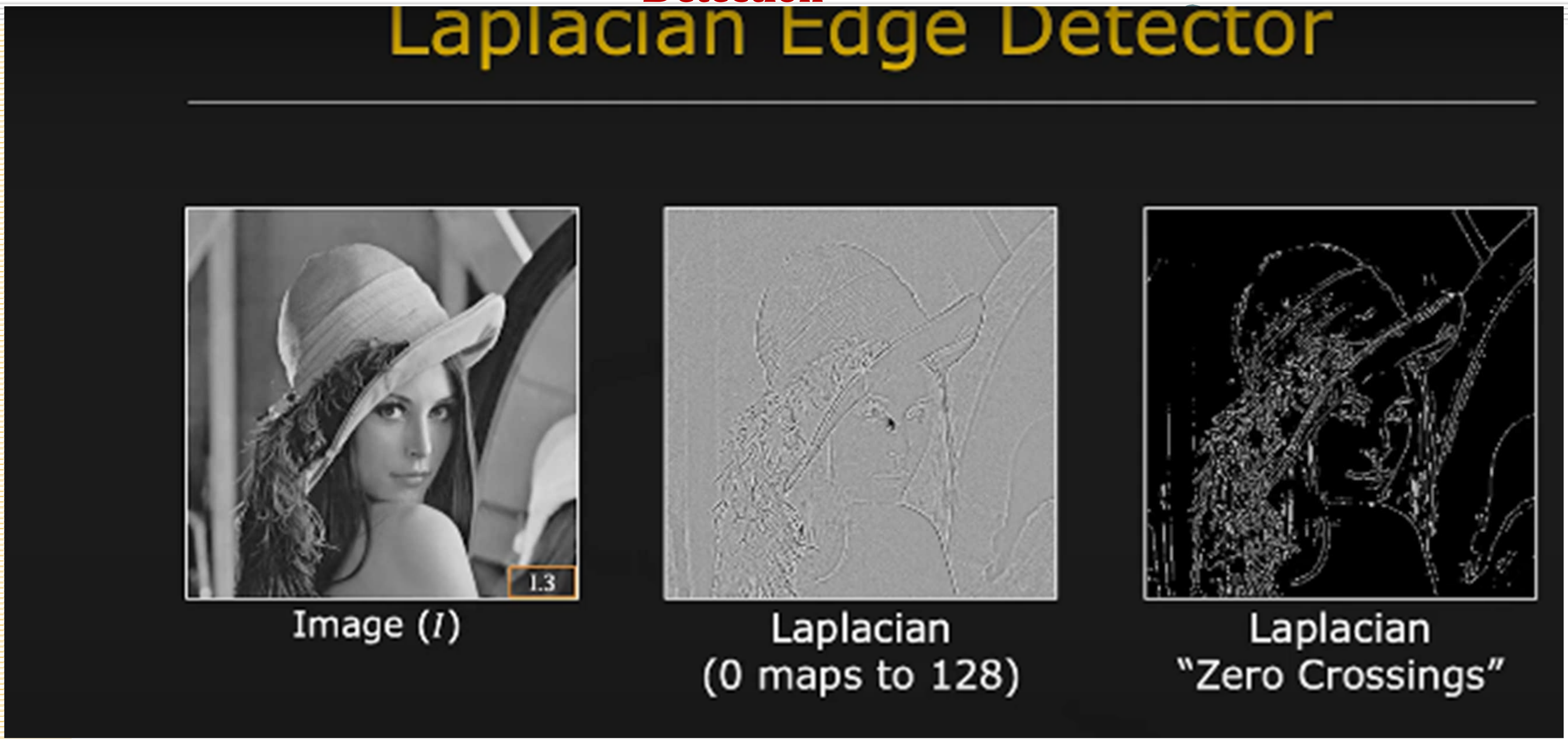
OR

-1	-1	-1
-1	8	-1
-1	-1	-1

When diagonal edges are included the Laplacian kernel will be

$$\frac{1}{6\epsilon^2}\begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & -20 & 4 \\ \hline 1 & 4 & 1 \\ \hline\end{array}$$

Effect of Noise on Edge Detection – Merging Gaussian Filter with Edge Detection



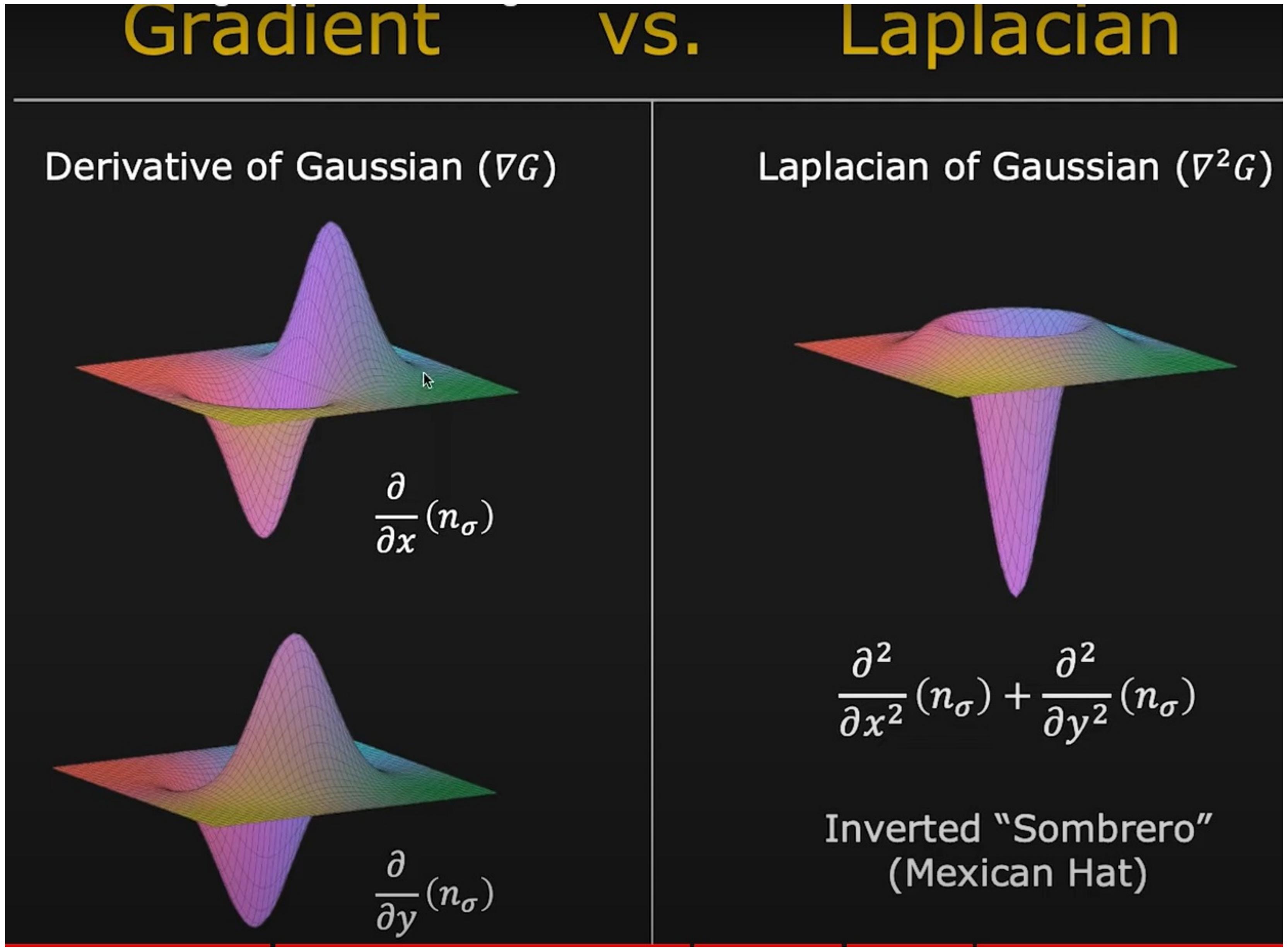
Convolution using Gradient-on-Normal & Laplacian-on-Normal Operator:

-1	-2	0	2	1
-2	-3	0	3	2
-3	-5	0	3	5
-2	-3	0	3	2
-1	-2	0	2	1

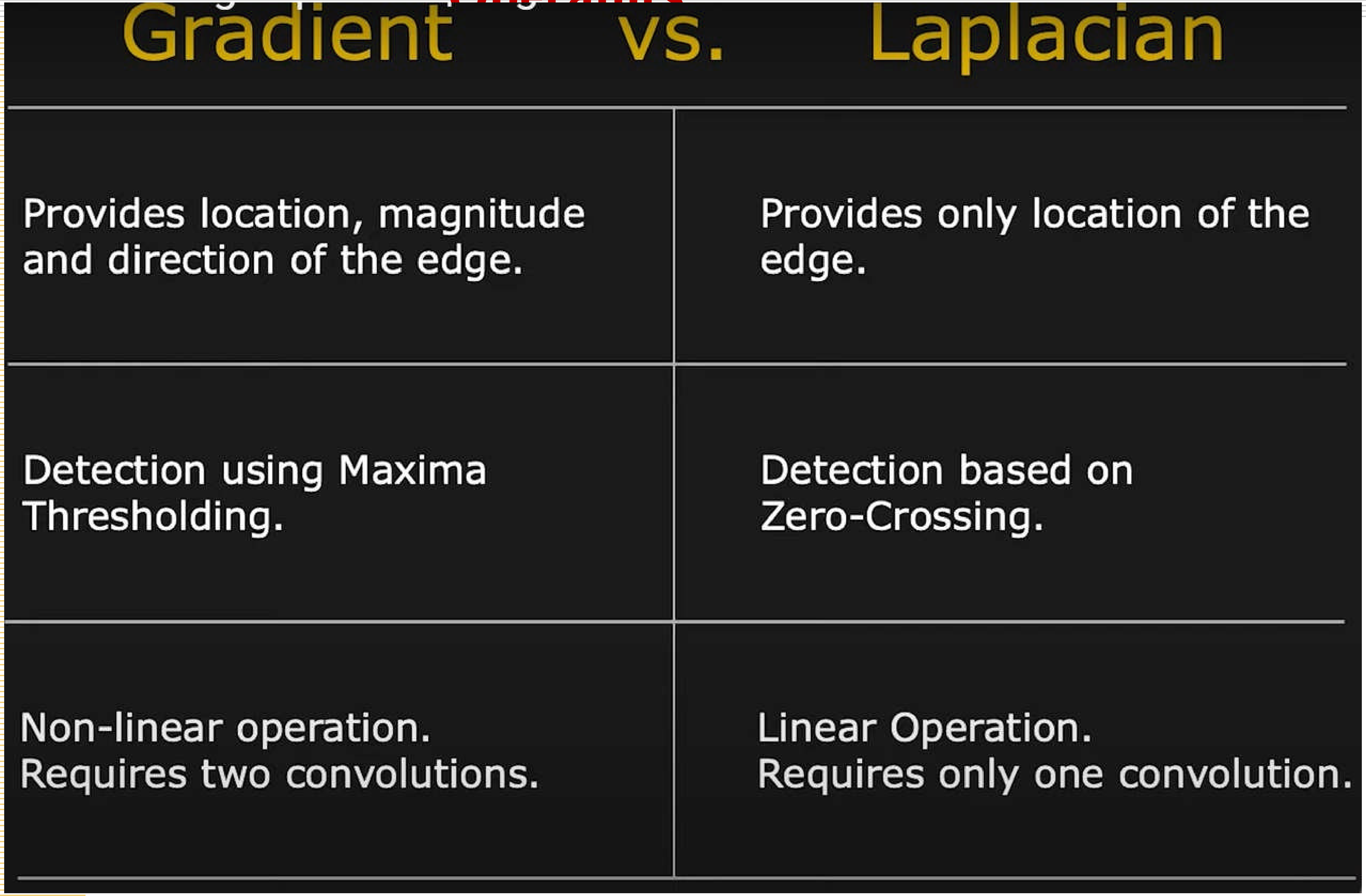
Similar to
Derivative of Gaussian

1	1	1
1	-8	1
1	1	1

Similar to
Laplacian of Gaussian


Gradient vs. Laplacian
Derivative of Gaussian (∇G)
Laplacian of Gaussian ($\nabla^2 G$)
$\frac{\partial}{\partial x}(n_\sigma)$
$\frac{\partial}{\partial y}(n_\sigma)$
$\frac{\partial^2}{\partial x^2}(n_\sigma) + \frac{\partial^2}{\partial y^2}(n_\sigma)$
Inverted "Sombrero" (Mexican Hat)

Comparison of Gradient and Laplacian Operators:

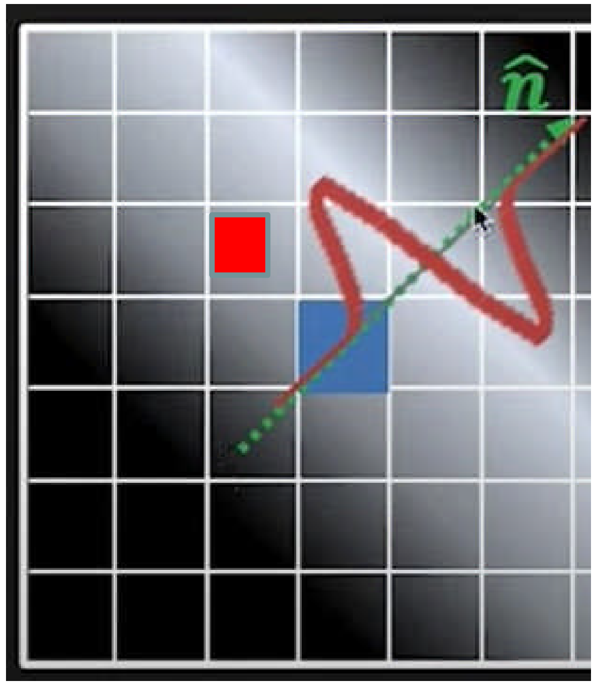


Gradient	Laplacian
Provides location, magnitude and direction of the edge.	Provides only location of the edge.
Detection using Maxima Thresholding.	Detection based on Zero-Crossing.
Non-linear operation. Requires two convolutions.	Linear Operation. Requires only one convolution.

Two convolution operations and non-linear operations to find the magnitude and direction

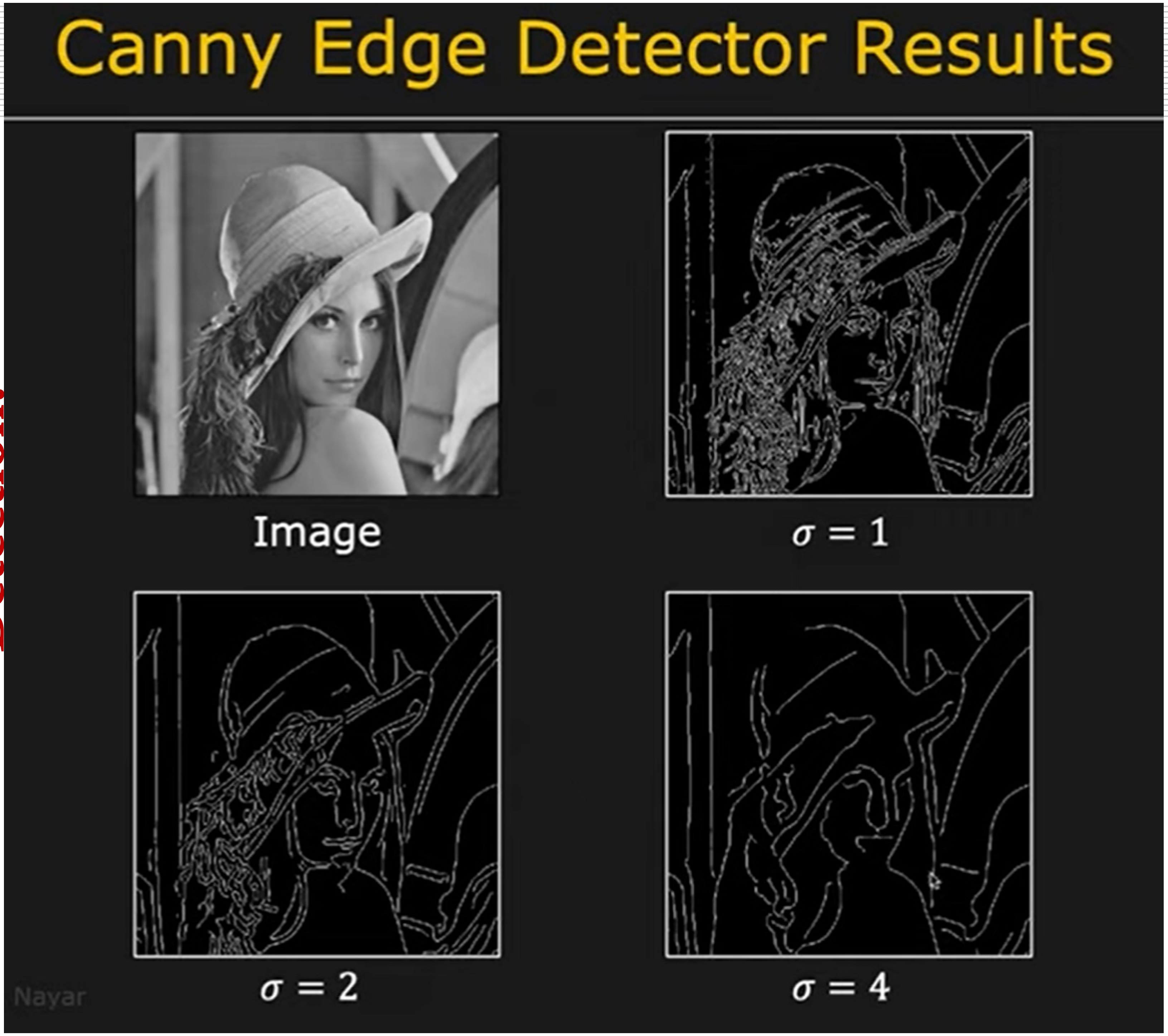
Canny Edge Detector:

- Canny Detector uses the best properties of gradient as well as Laplacian operator. It is the most popular edge detector.
- Smooth the image: $\mathcal{N}_\sigma * I$ convolving Image with Gaussian filter
- Compute Image gradient (2-components, in the direction of y- and x-axes) using Sobel filter $(Gaussian\ or\ 3 \times 3\ or\ 5 \times 5)$: $\nabla \mathcal{N}_\sigma * I$
- Find gradient magnitude at each pixel: $\| \nabla \mathcal{N}_\sigma * I \|$
- Find gradient orientation at each pixel: $m = {}^{\nabla \mathcal{N}_\sigma * I} / _{\| \nabla \mathcal{N}_\sigma * I \|}$
- One directional Laplacian (second derivative) along the gradient direction at every pixel: $\frac{\partial^2 (\mathcal{N}_\sigma * I)}{\partial\, m^2}$
- The Effect of σ the spread of Gaussian and its choice.
  Small σ will extract edges of high resolution.
  Large σ will extract edges at lower resolution.
  Thus, resolution scaling can be done by changing σ.

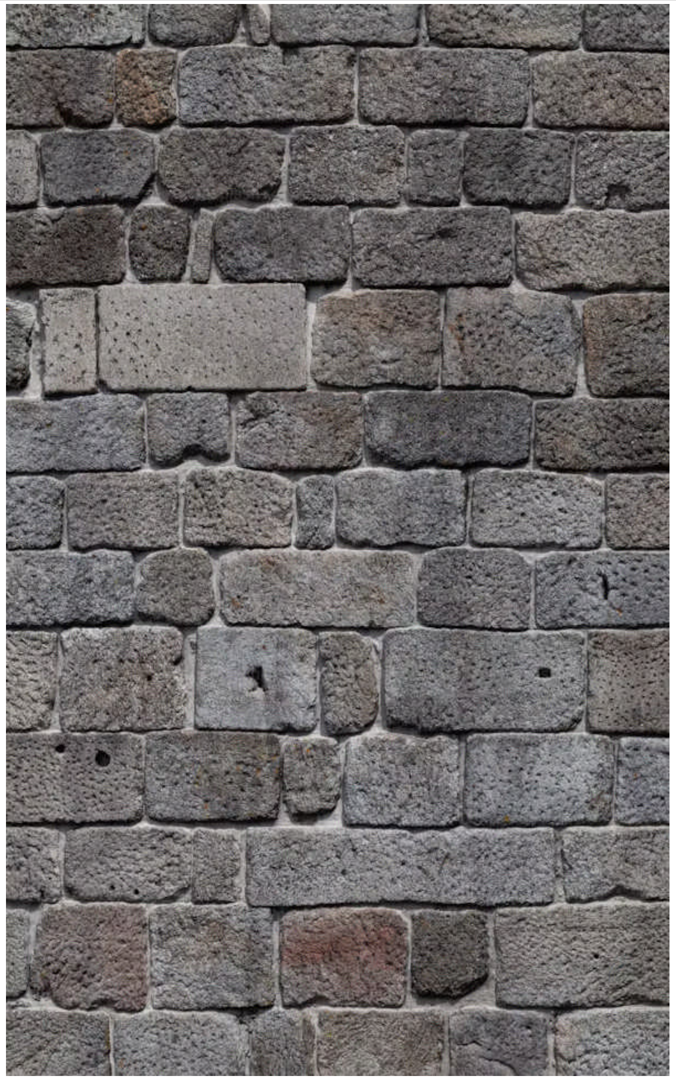


$\| \nabla \mathcal{N}_\sigma * I \|$ is the brightness

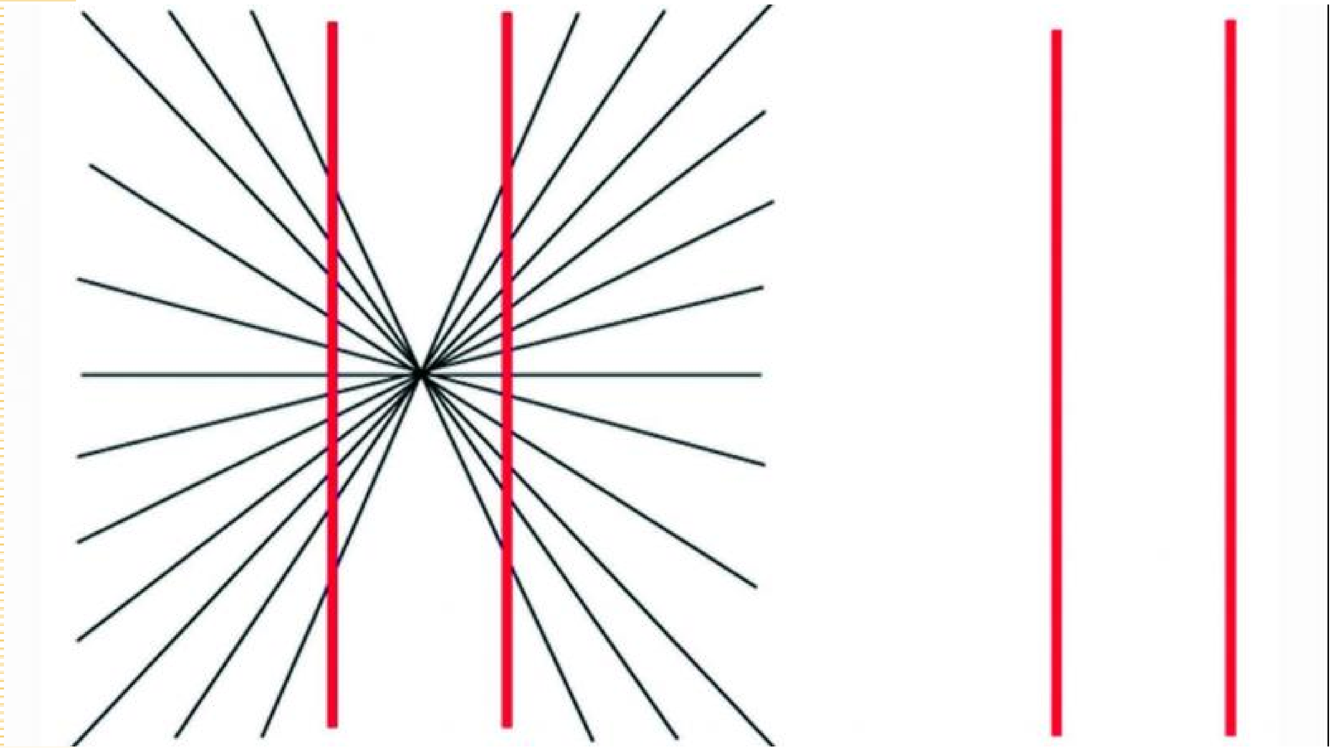
Canny Edge Detection:



Canny Edge Detector Results

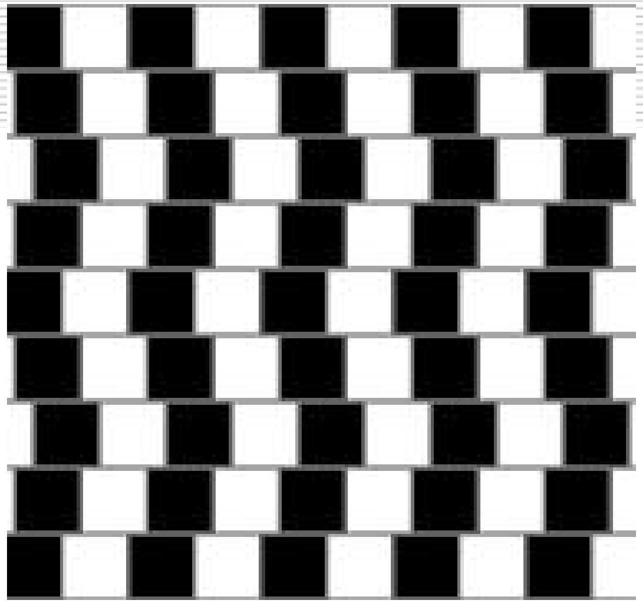


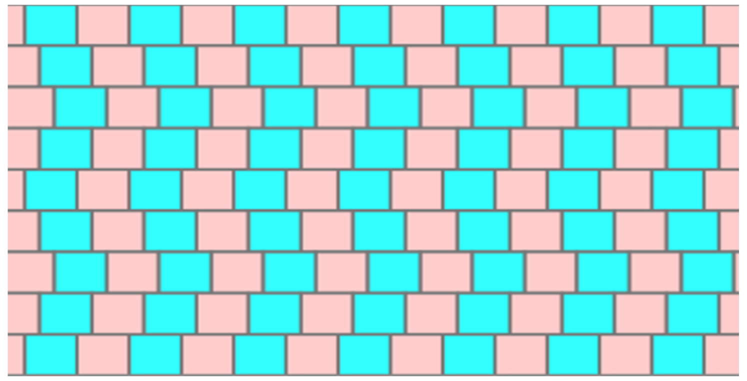
Perception and Illusion:



Hering edge illusion
Edward Hering 1861

We see acute angles as less acute.





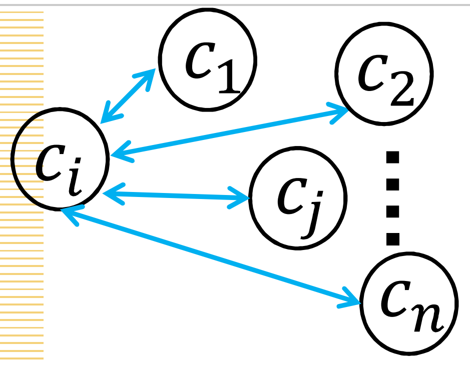
Café wall illusion
A. H. Pierce 1898

Reason: We see black squares smaller than they actually are.

Diffusion and Image Enhancement

- Fundamentals:
  - Diffusion and Laplacian
  - Graph Laplacian
  - Laplacian of a Function
- Graph Laplacian:
  - Network (graph) Clustering
  - Matrix Rank Reduction and Matrix Completion Problems
  - Miscellaneous applications of Diffusion on Networks
- Human and Computer vision:
  - Edge Detection and Laplacian Filter
- Diffusion and Image Enhancement
- Diffusion and Image Generation

Diffusion on a Graph and Application to Image



$$\frac{\partial \vec{c}}{\partial t} = \rho[A\vec{c}] - \rho[D\vec{c}] = \rho[A - D]\vec{c}$$

$$\frac{\partial \vec{c}(t)}{\partial t} = \alpha c;\ \vec{c}(t) = \vec{c}(0)e^{-\alpha t}$$

$$\frac{\partial \vec{c}}{\partial t} + \rho[D - A]\vec{c} = 0;\ \frac{\partial \vec{c}}{\partial t} + \rho[L]\vec{c} = 0;$$

$$When\ the\ system\ converges\ \frac{\partial \overrightarrow{c_\infty}}{\partial t} = 0;\ \rho[L]\overrightarrow{c_\infty} = [L]\overrightarrow{c_\infty} = 0;$$

A and D are symmetric. [D - A] is symmetric. All elements of L are REAL.
$[D - A]\ is\ positive\ definite\ or\ positive\ semidefite;$
$All\ Eigenvalues\ are\ non - negative\ reals. Eigenvectors\ orthogonal.$

As every row of [D - A] sums up to 0, there is a 0 eigenvalue, and corresponding eigenvector is a vector with all elements equal.

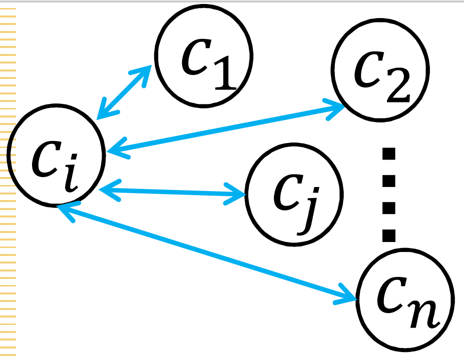
$\overrightarrow{c_\infty}$ is that eigenvector and corresponding eigenvalue = 0;

The other eigenvectors are interesting, partitions of a network (graph).

The method is known as FIELDER's "Spectral Graph Partitioning."

Spectral Graph Partitioning is different from Matrix Spectral Analysis.

Diffusion and Image Enhancement



$$\frac{\partial \vec{c}}{\partial t} = \rho[A\vec{c}] - \rho[D\vec{c}] = \rho[A - D]\vec{c}$$

$$\frac{\partial \vec{c}}{\partial t} + \rho[D - A]\vec{c} = 0; \quad \boxed{\frac{\partial \vec{c}}{\partial t} + \rho[L]\vec{c} = 0;} \qquad (1)$$

A and D are symmetric. [D - A] is symmetric. All elements of L are REAL.
$[D - A]$ *is positive definite or positive semidefite;*
All Eigenvalues are non − negative reals. Eigenvectors orthogonal.

Eigen − vectors of Laplacian matrix are $\vec{v_i}$ ***for*** $i = 1$ ***to*** n

$\vec{v_i}$***s are orthogonal and span the*** n ***− dimensional space***

$$\vec{c} = \sum_{i-1}^{n} a_i(t)\,\vec{v_i}$$

$$Then, \sum_{i=1}^{n} \frac{da_i}{dt}\vec{v_i} + \sum_{i=1}^{n} \rho a_i L\vec{v_i} = \sum_{i=1}^{n} \frac{da_i}{dt}\vec{v_i} + \sum_{i=1}^{n} \rho a_i \lambda_i \vec{v_i} = \boxed{\sum_{i=1}^{n}\left(\frac{da_i}{dt} + \rho a_i \lambda_i\right)\vec{v_i} = 0}$$

Diffusion and Image Enhancement

$\boxed{\sum_{i=1}^{n}\left(\frac{da_i}{dt}+\rho a_i\lambda_i\right)\vec{v_i}=0}$; $\textbf{As } \vec{\mathrm{v_i}}\textbf{ s are orthogonal, for all } i$, $\boxed{\frac{da_i}{dt}+\rho a_i\lambda_i=0}$

Solutions for these ODEs are $a_i(t)=a_i(0)e^{-\rho\lambda_i t}$; *and* $\boxed{\vec{c}(t)=\sum_{i=1}^{n}a_i(0)e^{-\rho\lambda_i t}\,\vec{v_i}}$ (2)

$\boxed{a_i(0)=\frac{\vec{c}(0)\cdot\vec{v_i}}{|\vec{v_i}|^2};}$ As $\vec{c}(0)$, and Eigenvectors $\vec{v_i}s$ of L are known, $a_i(0)$s are known. We can solve $\vec{c}(t)$, Eq. (2)

From Eq (2), $\vec{c}(t=\infty)=\mathbf{0}$ for all λ_i except $\lambda_1=0$. So, $\vec{c}(t=\infty)=a_1(0)\vec{v_1}$

As all elements of $\vec{v_1}$ are equal, and $\vec{v_1}$ is normalized, $\boxed{\vec{c}(t=\infty)=\left[\frac{c_1(0)+\cdots+c_n(0)}{n}\right]\vec{1}}$

For a connected graph, all nodes will have the same share of the initial fluid content.

Heterogenous Diffusion and Image Enhancement

$\boxed{\frac{\partial \vec{c}}{\partial t} + \rho[L]\vec{c} = 0; \quad (1);}$ As $t \to \infty$; $\frac{\partial \vec{c}(t)}{\partial t} \to 0$; $[L]\vec{c} = 0$; $(D - A)\vec{c}(t = \infty) = 0$

$\boxed{\vec{c}(t = \infty) = D^{-1}A\,\vec{c}(t = \infty)}$ D is invertible if the graph is connected.

$\vec{c}(t = \infty) = W\,\vec{c}(t = \infty); where\; W = D^{-1}A$

As a linear or recursion equation:

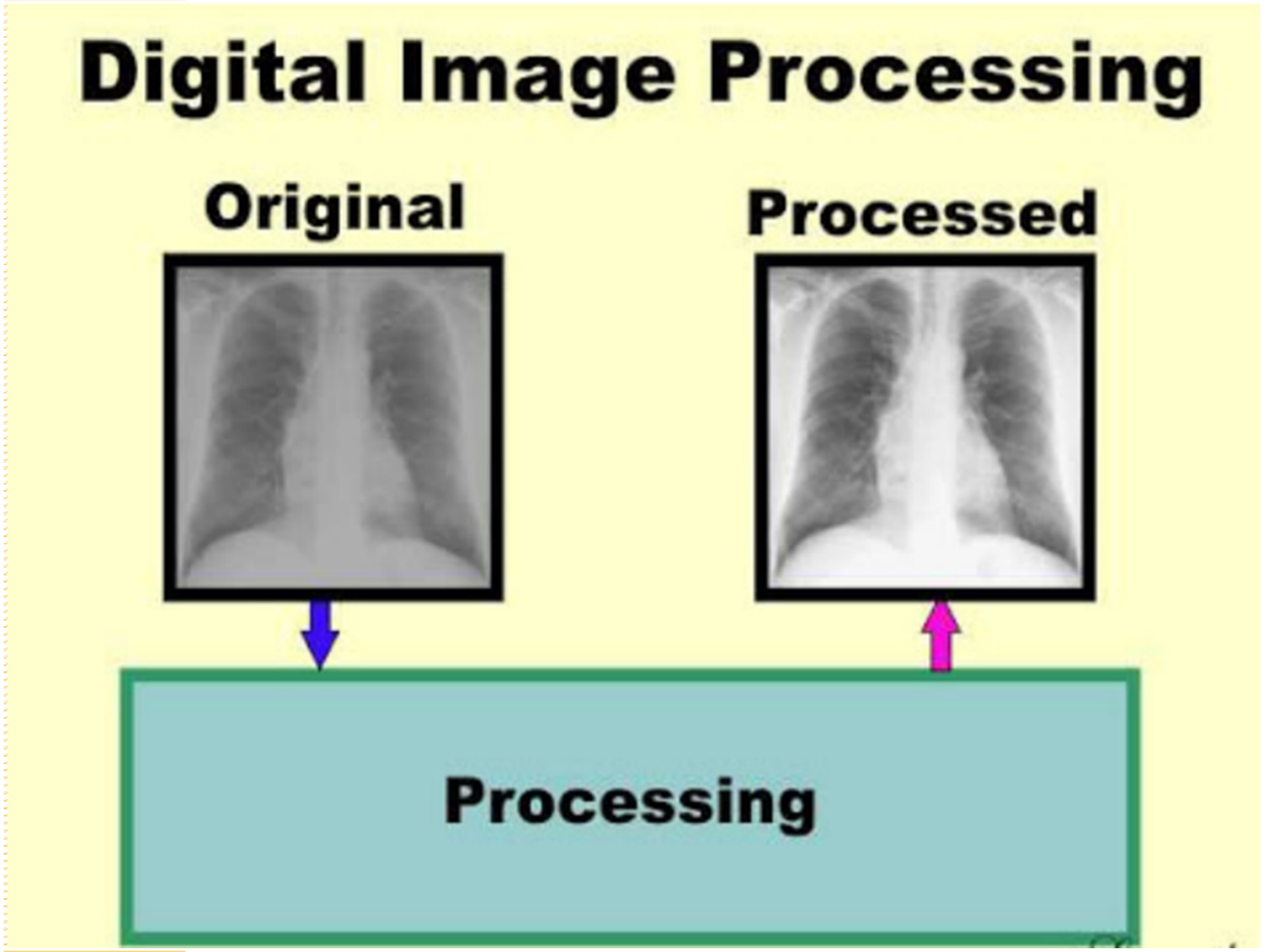
$\boxed{\vec{c}(t = k + 1) = W\,\vec{c}(t = k);}$ (3)

Heterogenous Diffusion and Image Enhancement

$$\boxed{\vec{c}(t = k + 1) = W\,\vec{c}(t = k);} \quad (3)$$

1. W is low rank for a meaningful image and high rank for random noise. (**Please remember embedding dimension – Diffusion Map**).
2. In a Heterogenous network, each edge has its own diffusion rate. $\boldsymbol{A}$ is still symmetric. Sum of i^{th} row is the degree of i^{th} node.
3. Each pixel of the image is the network node.
4. There is a affinity between a pair of nodes, which determines the connection weight.
   - The affinity could be the grey levels (or RGB levels) of the pixels.
   - It could be the proximity between two nodes.
   - Original image is k=0; We recursively change the image using Eq. (3)

Heterogenous Diffusion and Image Enhancement



When the affinity between a pair of pixels is the grey level of the two pixels, then the process of diffusion will make similar pixels more similar –

The result is a sharper image.

Please note that at every step the pixel grey level changes necessitating calculation of the affinity matrix at every step.

When the proximity between a pair of nodes is the affinity, the process of diffusion is results in a smoother image. The affinity matrix in this case is static.

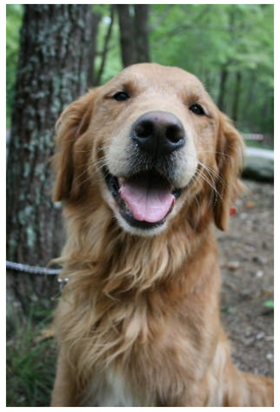
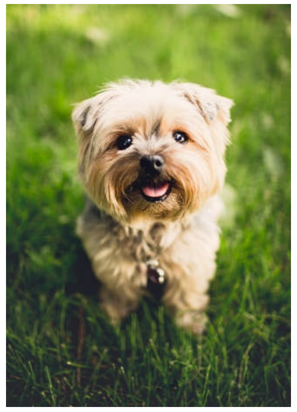
In practice a combination of the two affinity matrices is used, like Gaussian filter in combination with Laplacian filter.

We can use Gaussian kernel with the distance affinity matrix on a high dimensional data. . hen the affinity between a pair of pixels is

Diffusion Map and Dimensionality Reduction

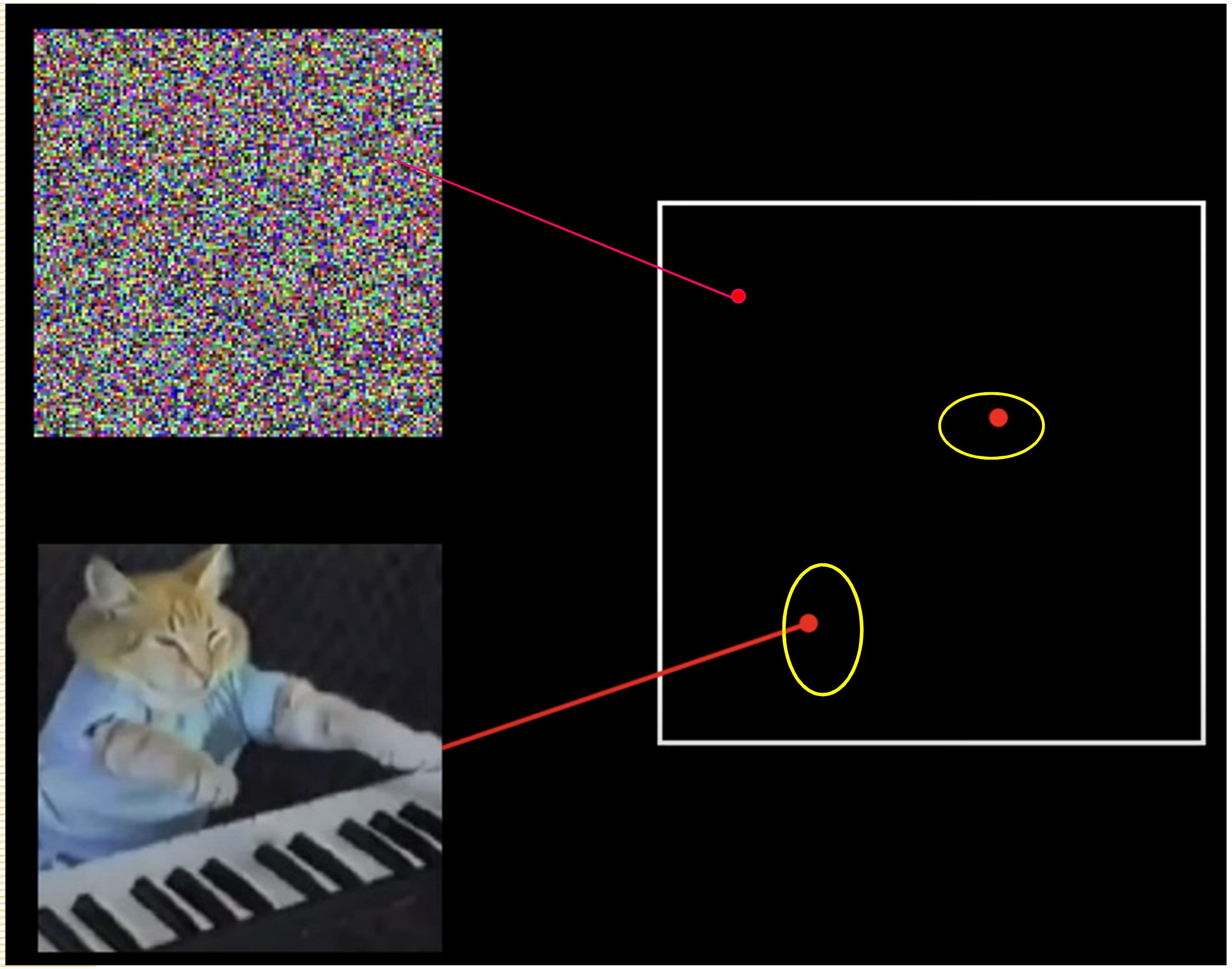
1. Embedding dimension, Riemannian Geometry, Geodesics.
2. A few Dimension Reduction and 2-D visualization Tools.
3. Principal Component Analysis (PCA).
    1. Multidimensional Scaling (MDS). $Strain(D_x, D_y) = \left\| J^T \left(D_X^2 - D_Y^2\right) J \right\|_F^2$
    2. Isometric Feature Map (Isomap) – Same as MDS but uses geodesic distances in high dimension, Using a method similar to DBSCAN. It assumes that for close data points, Euclidean distances are same as geodesic distance. Then add up considering neighboring data.
4. Diffusion Map: Find the connectivity using Euclidean distance, using a Gaussian kernel. Use it as probability of moving from one data point to other. Run the diffusion process as a random walk Markov process till convergence. We get the diffusion map.

Diffusion Models for Image Generation

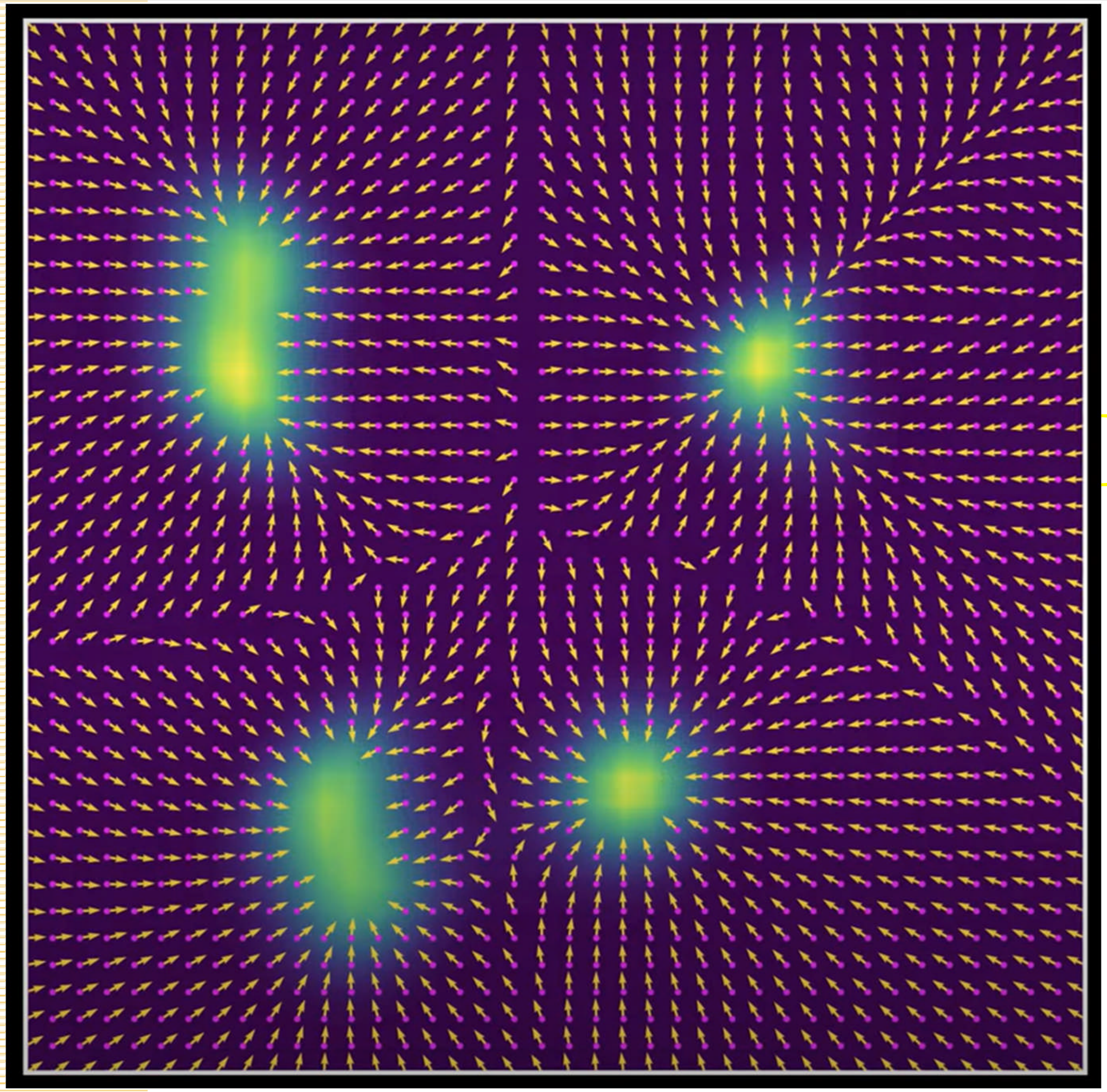
100 X 100 color image space have $10000^{256\times256\times256}$ possibilities. All “Apple”, “Dog”, “Face” etc. are only a subset of the whole possible set of points. If we plot them on a two dimensional plane, Apple images, dog images, face images etc. will form clusters, because different images will have similar pixel value distribution. A vast space will look as random noise images as shown on the right.

Diffusion Models for Image Generation



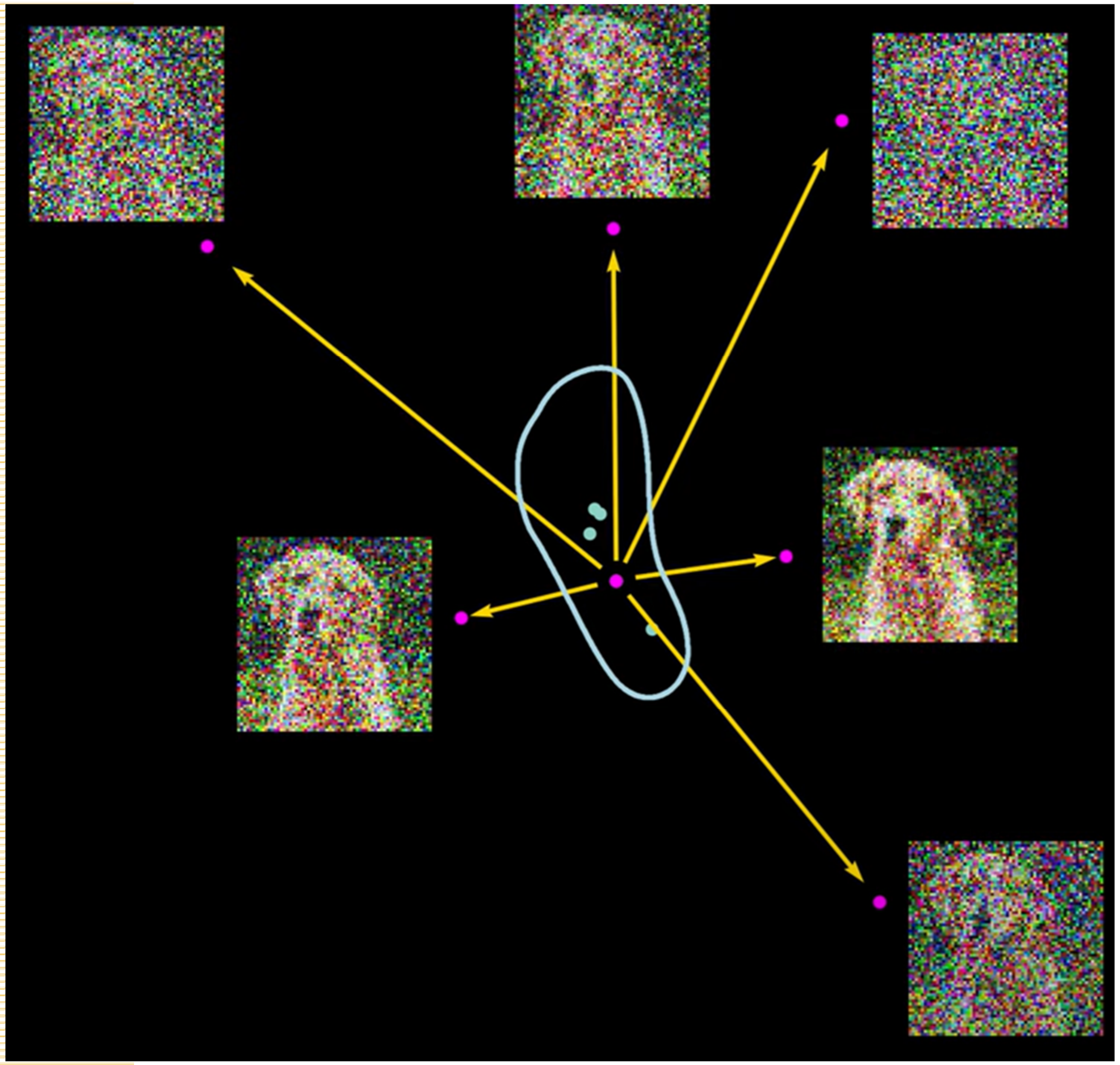
Vast area of the image space will be points corresponding to noise images. There will be small clusters of meaningful images, like cats, cars, dogs, human face, etc. Let us restrict the random images Gaussian distribution.

Diffusion Models for Image Generation



The blue hills represent different meaningful images, like dogs, cats, cars. The red dots are image points which are random noise images. Starting at one of the red points, the diffusion model learns the direction of the convergence arrows leading to one of the meaningful image hill. Starting at different random points, we end up at different images, and the converged point on the hill, more often than not, will give a new image, not one of those which were used during the training.
Comparison with Hopfield model – Hopfield network stores a set of fixed images. From an initial random pixel distribution, it will converge to a fixed image – not a variation of it.

Diffusion Models for Image Generation



From a meaningful image we generate noisy images, adding Gaussian noise, in the different directions. We learn how to return to the original image, in a number of steps (usually 1000 steps).

We repeat this learning for all sorts of images we want to generate. Thus, the whole space of image points are get learnt. Note that, for the learning we do not need any annotated image. We add Gaussian noise, in steps, to get the training sample pair – noisy input clean output.

After sufficient learning over the image space, we start from any random point and the diffusion model converges on a meaningful image, though not exactly one of which are used for the training.

- Thank you for your attention.
  - Questions are welcome.