Lecture 6 (contd)

* Gaussian filter

give higher weight to pixels near the center

$$\frac{1}{273} * \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

2D gaussian $G_\sigma(x,y) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$

Separable implementation

$$I_G = I * G = \sum_i \sum_j I(i,j)\, e^{\frac{-i^2+j^2}{2\sigma^2}}$$

$$= \underbrace{\sum_i e^{\frac{-i^2}{2\sigma^2}}}\ \underbrace{\sum_j I(i,j)\, e^{\frac{-j^2}{2\sigma^2}}}$$

$$= (I * G_y) * G_x = I * G_x * G_y$$

Complexity of separable implementation

$M \times N$ image

$m \times m$ filter

One 2D pass : $M \times N \times m^2$ operation

Two 1D pass : $2 \times M \times N \times m$ operations

$$2(m) \cdot MN < MN(m)^2$$

repeated application of gaussains :

$$I * G_{\sigma_1} * G_{\sigma_2} = I * G_{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

$$2 \times (M \times N \times m^2) < (M \times N \times (2m)^2)$$

? Selecting gaussiance variance (σ)

high the sigma $\rightarrow$ bigger the filter

rule is to us filter $m = 5\sigma \Rightarrow \sigma \leq \frac{m}{5}$

$\frac{1}{2.5}$ [0.14 | 0.6 | 0.6 | 0.14] $\cdot a = \frac{5}{6}$

$\frac{1}{16}$ [1 | 4 | 6 | 4 | 1] $a = 1$

* Median filter :

$I_{med}(i,j) = \text{median}\{ I(x,y) \mid (x,y) \in \text{with } (i,j)\}$

5	5	99
6	2	4
7	97	3

→ 2, 3, 4, 5, 5, 6, 7, 97, 99

↑

med val = 5

Good for salt & pepper noise.

* Multiple scale analysis :

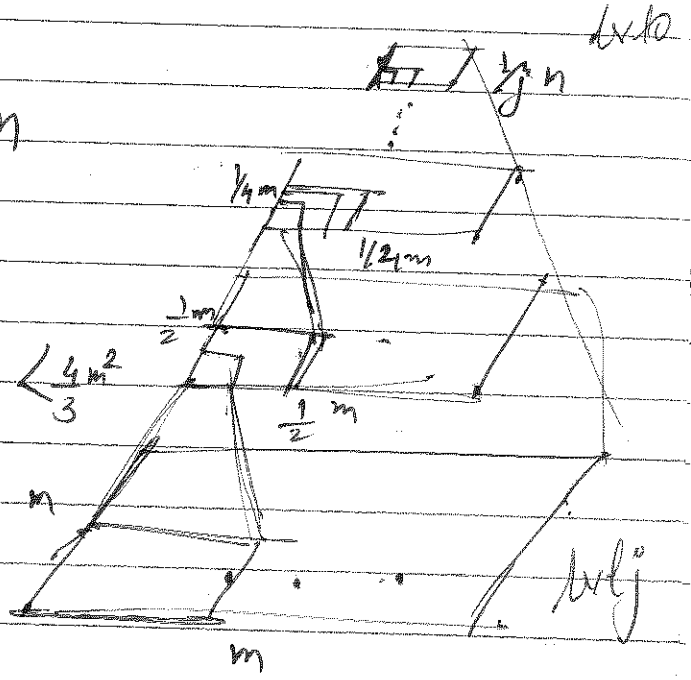
analyse image with different filter size

Image pyramid:

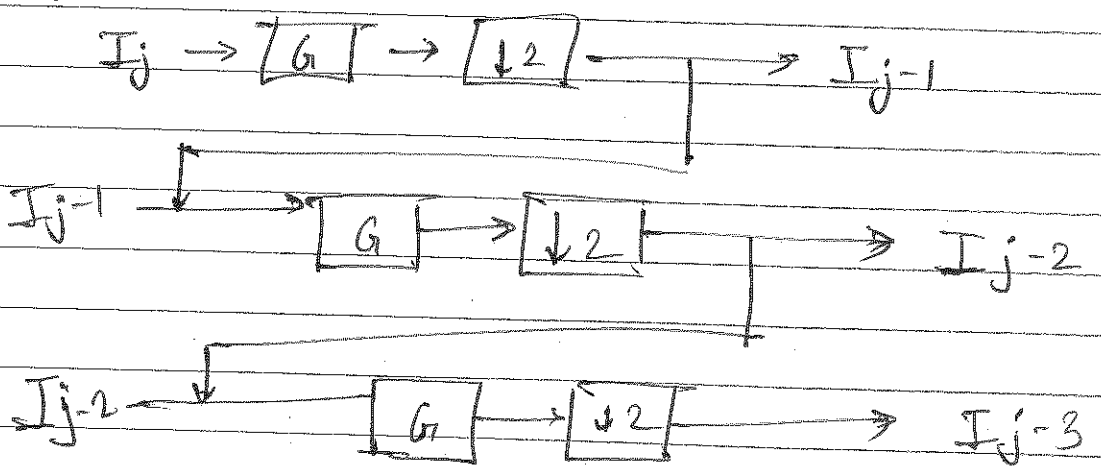
$m = 2^J \Rightarrow J = \log_2 m$

total # pixels =

$m^2 + \frac{1}{4} m^2 + \frac{1}{16} m^2 + \ldots < \frac{4m^2}{3}$



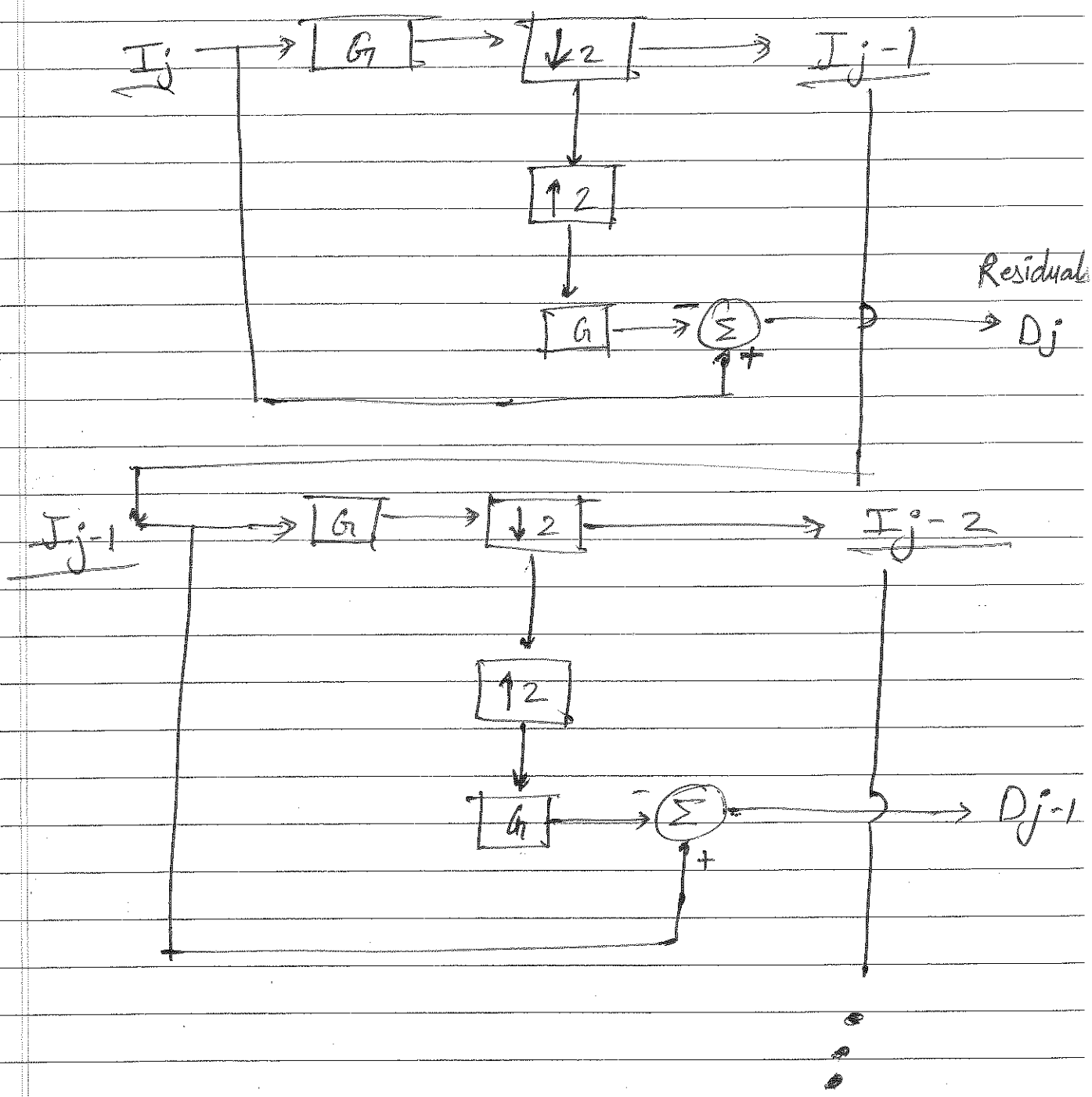
Gaussian pyramid:



- Helps to tackle Aliasing images
- useful for analysis

Laplacian pyramid :

- more use for image compression
- variance is smaller



Lecture 7

☆ Feature Detection

anything like eyes, hair, etc.

Two approach:
1. Feature engineering
2. Deep learning

1. Feature engineering

used when problem is known or available

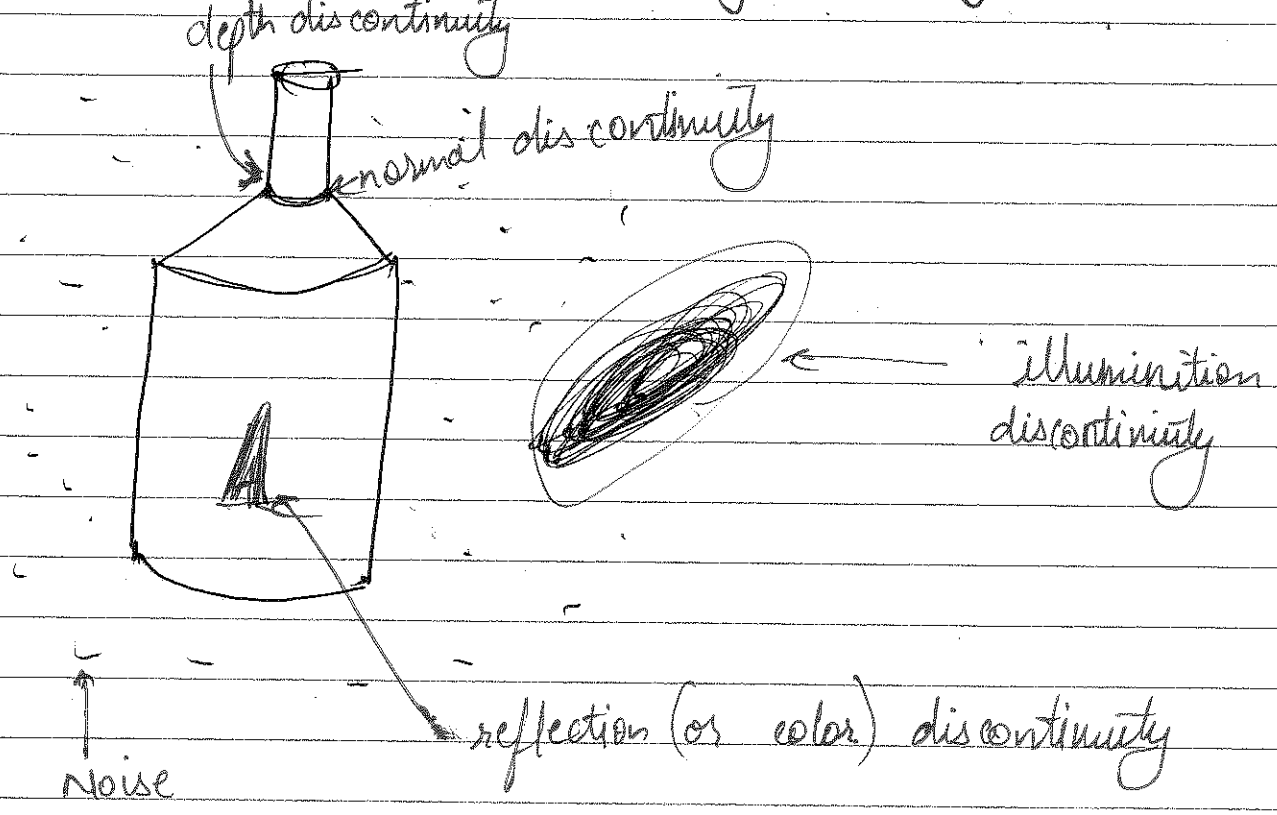
* Feature types
  a) global: eg color distribution (histogram)
  b) local: eg edges or corners

* goal:
  - characterize images
  and/or
  - local neighborhoods

1.1. Edge detection

requirements:
- correspond to scene elements
- invariant [illumination, pose, viewpoint, scale]
- reliable detection

edge ⇒ location with change in image

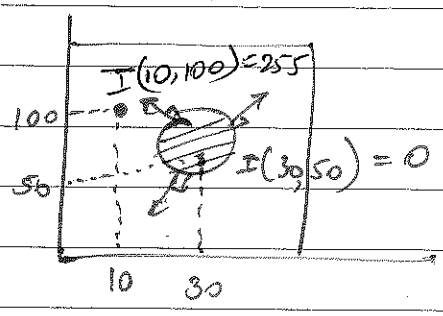


* edge detection steps:

1) Smooth image to reduce noise (without affecting edges.)
2) Enhance edges
3) Detect edges
4) Localize edges.

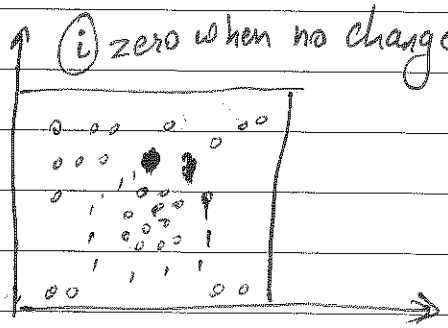
Image gradiant

Image : $I(x,y)$



$$\nabla I(x,y) = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

$$= \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$



Note : image gradiant vectors are $\perp$ to the edges.

$$\frac{\partial I}{\partial x} = \lim_{x \to 0} \frac{I(x+h,y) - I(x,y)}{h}$$

$$\frac{\partial I}{\partial y} = \lim_{x \to 0} \frac{I(x,y+h) - I(x,y)}{h}$$

$h = 1$

$$\frac{\partial}{\partial x} I(x,y) = I(x+1,y) - I(x,y).$$

edgle ⇒ edge element

magnitude : $|\nabla I| = \sqrt{x^2+y^2}$

angle : $\theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$

* Edge detection :

$$E(i,j) = \begin{cases} 1 & \text{if } |\nabla I(i,j)| > \tau \\ 0 & \text{otherwise} \end{cases}$$

(τ : Threshold)

(E : Edgemap)

• So,

$$\frac{\partial I(x,y)}{\partial x} = I(x+1, y) - I(x,y)$$

$$\Delta x = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

y → [-1 | +1]
x x+1

← detects vertical edges

similarly,

$$\Delta y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

* Central difference

$$\frac{\partial}{\partial I} I(x,y) = \lim_{h \to 0} \frac{I(x+h,y) - I(x-h,y)}{2h}$$

here $h = 1$,

$$= I(x+1,y) - I(x-1,y)$$

-1	0	1
$x-1$	x	$x+1$

$$\Delta x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ Smooth

derative

$$\Delta y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$ smooth, deri

Note Explanation,

$$\frac{\partial}{\partial I} I(x,y) = (+) I(x+1, y) + 0 - I(x-1, y)$$

$(+)$ $x+1$ $+0$ $(-)$ $x-1$

$$\downarrow$$

$$(-_{x-1} \quad 0 \quad +_{x+1})$$

$$\downarrow$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

*Implementation using –

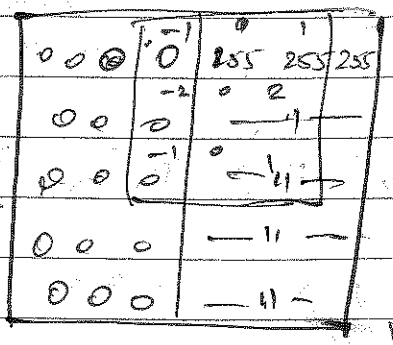
* Sobel filter : (open cv)

noise reduction ↓ edge detection ↓

$$\Delta x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\Delta y = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

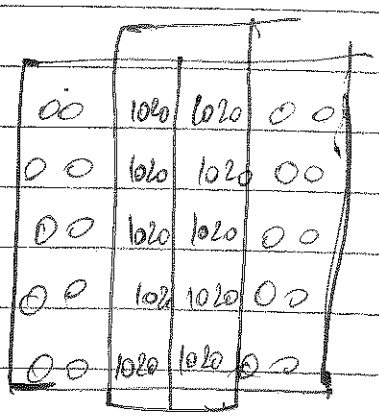
Example:



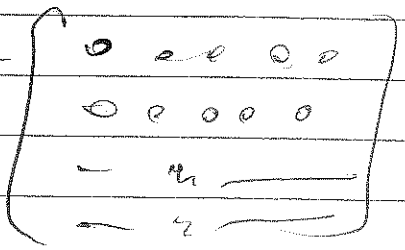
$= 1 \times 255 + 2 \times 255 + 1 \times 255$

$= 1020$

$\Delta x =$

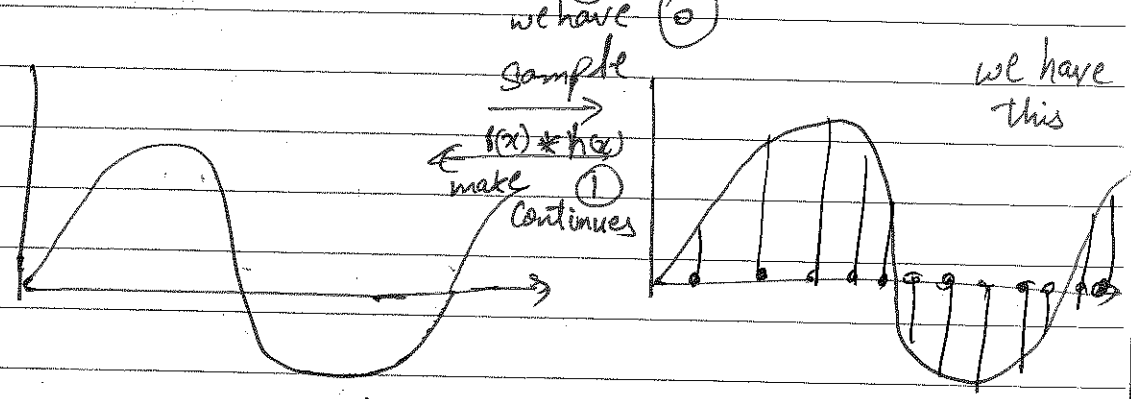


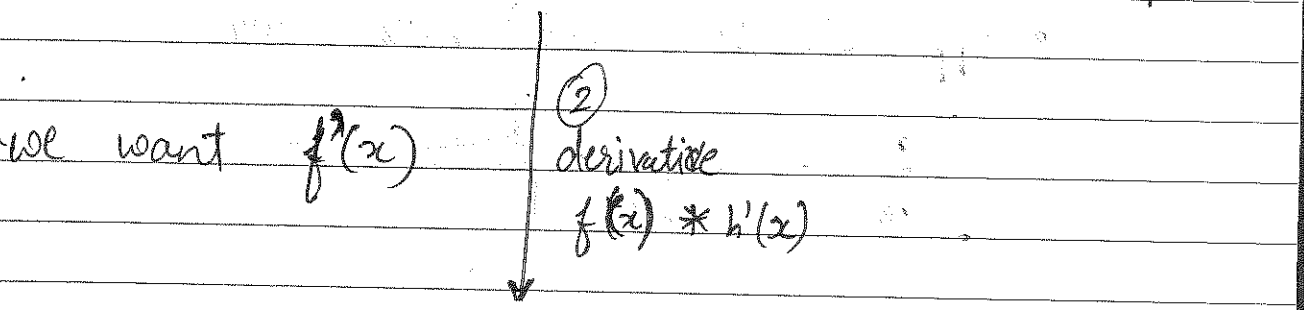
$\Delta y =$

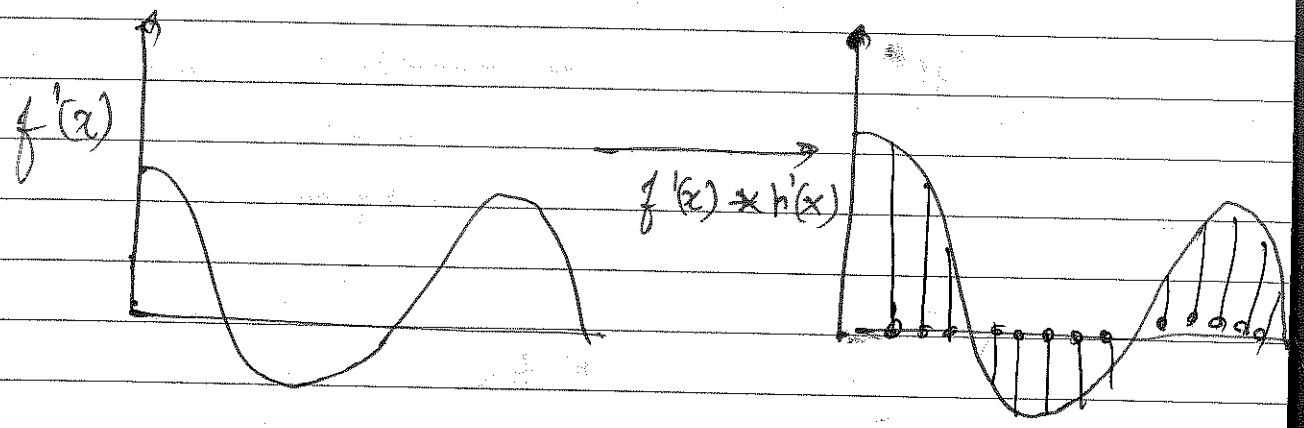


*Implementation using –

- More accurate derivative (using 1D) explained :





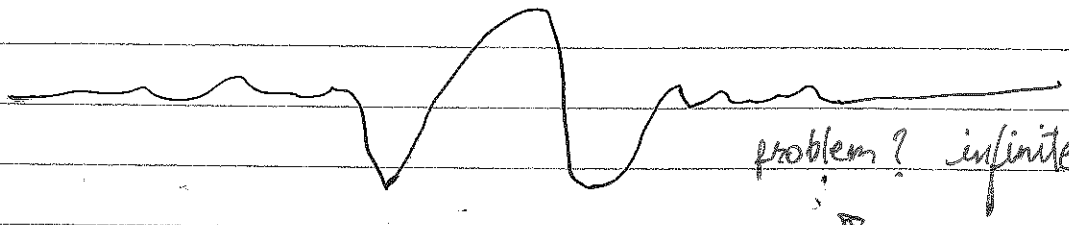


So we have to define $h(x)$, then convolve the continues func with derivative of $h(x)$ [$h'(x)$].

- Perfect interpolator

under certain conditions perfect reconstruction.

$$h(x) = \text{sinc}(x) = \frac{\sin(\pi x / T)}{(\pi x / T)}$$



problem? infinite tail!

- approximate $h(x)$ with $G(x)$

$$f_s \geq 2 f_m .$$

sampling frequency (pixel/mm)

variation
less variation then less f_m
$\uparrow f_m = \uparrow$ variation
$\downarrow f_m = \downarrow$ variation

$$I_x = I * G_x$$
$$I_y = I * G_y \qquad f[x] * g[x]$$

using separable property of gaussian

$$I_x = I * G'[x] * G[y]$$

$$I_y = I * G'[y] * G[x]$$

I_x = [image] $*$ [derivative gaussian] $*$ [smooth]

$I_x = I * G'[x] * G[y]$

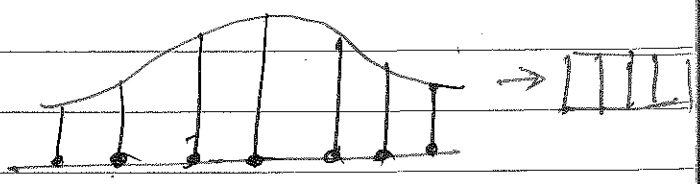
$I_y = I * G'[y] * G[x]$

I_y = [image] $*$ [ce qui vient] $*$ [smooth]

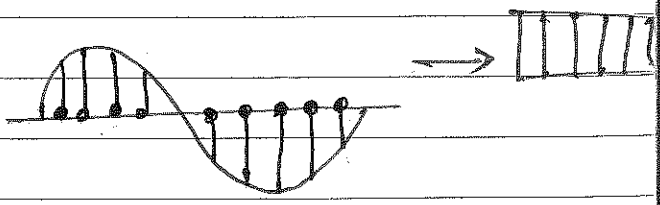
gaussian

So 1D gaussian,

$G_x = e^{\frac{-x^2}{2\sigma^2}}$

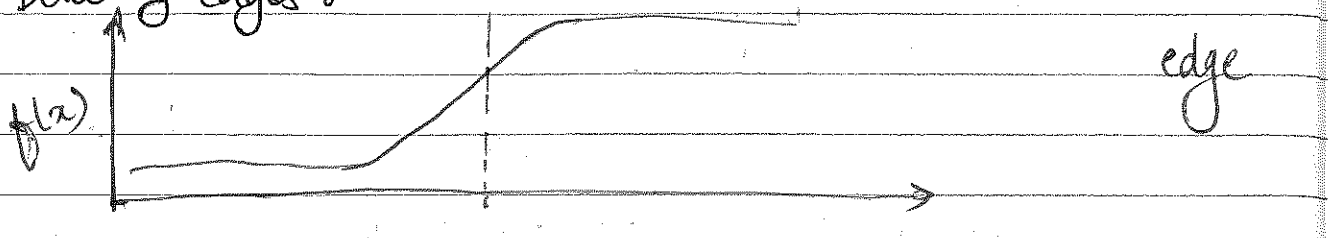


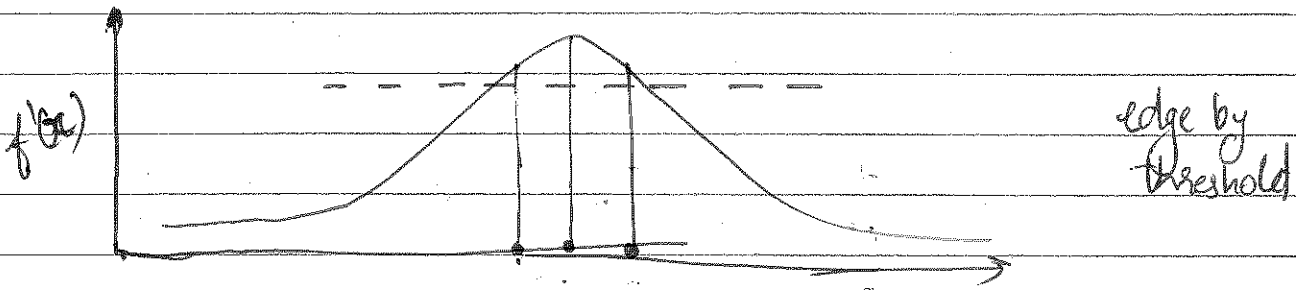
$G'_x = \frac{-x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}}$

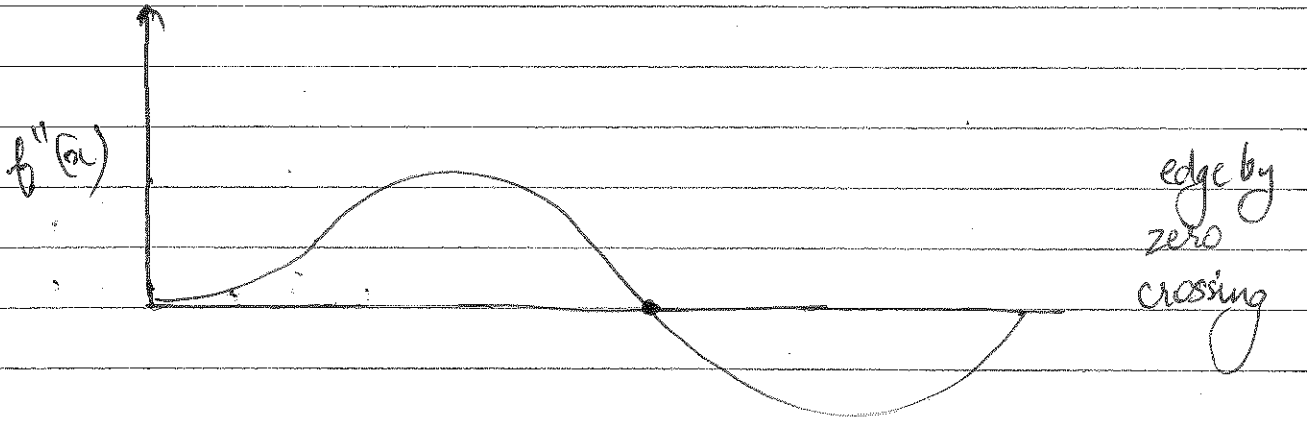


Lecture - 8 (feature detection contd.)

Detecting edges :







* In 2D, second derivative as scalar quantity :
Laplacian : $\Delta I = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = I_{xx} + I_{yy}$

~~$I_x = I(x+1) - I(x)$~~

$I_{xx} = (I(x+1) - I_x) - (I(x) - I(x-1))$

$= I(x+1) - 2I(x) + I(x-1)$

$\Rightarrow$ [1 | −2 | 1]

$I_{yy} \Rightarrow$

1
-2
1

$I_{xx} + I_{yy} \Rightarrow$

0	1	0
1	-4	1
0	1	0

→ Laplacian filter

Apply this and look for crossing edges.

Smooth with gaussian before applying laplacian (bcoz 2nd degree derivative is more sensitive to noise)

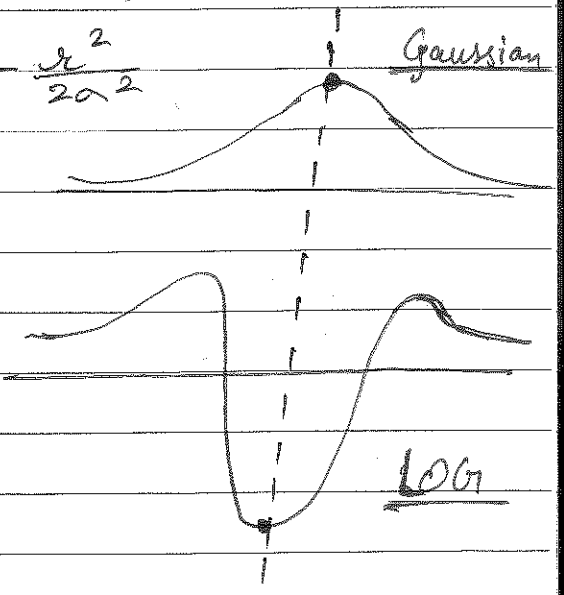
$$H = \nabla^2 (I * G) = \underbrace{\nabla^2 G} * I$$

(filter) LOG →
Lap of Gaussian

We know,

$$G = e^{-r^2/2\sigma^2} \qquad (r^2 = x^2 + y^2)$$

$$\nabla^2 G = \frac{r^2 - 2\sigma^2}{r^4} \; e^{-\frac{r^2}{2\sigma^2}}$$



* Edge detection using LOG:

1) Compute LOG: $H = (\nabla^2 G) * I$

2) Threshold: $E(i,j) = \begin{cases} 0 & \text{if } H(i,j) < 0 \text{ (output a binary image)} \\ 1 & \text{if } H(i,j) \geq 0 \end{cases}$

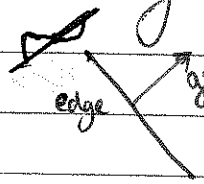
3) Mark the (zero crossing) edges at threshold (i.e $0 \to 1$ or $1 \to 0$)

LOG → Image = $\frac{L - \min(L)}{\max - \min} \times 255$

* Canny edge detection:

Detect edges at zero crossings of second order directional derivatives taken along the gradient. (2nd directional derivative)

n = gradient



if $|n| > T$ detect edges at zero crossing of $\frac{\partial^2}{\partial n^2}(I * G)$

- Comments:

1) attempts detect only if gradient magnitude is large enough ($|n| > T$)

2) smooth along edges to preserve edges.

3) Alternative to zero crossing of $\frac{\partial^2}{\partial n^2}(I * G)$ is

maximum at $\frac{\partial}{\partial n}(I * G)$.

- so lets see ~~how its done~~ what we have to do,

gradiant : $n = \nabla(I * G)$

directional derivative : $\frac{\partial (I * G)}{\partial n} = n \cdot \nabla(I * G)$

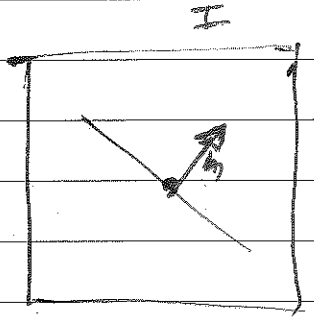
(n: direction, $\nabla(I * G)$: gradiant)

$n \cdot \nabla I * G = \nabla (I * G) \cdot \nabla (I * G)$

(n: direction of gradiant)

$= |\nabla (I * G)|^2$

mag of gradiant — so we find max of this and that our edge.



- Summary:

If $|\nabla (I * G)| > \tau$ set edge at max of $|\nabla (I * G)|$ along the direction of $\nabla (I * G)$

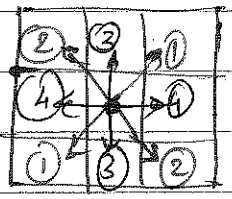
* Additional components:

1) Non-maximum suppression

Need: local maximum of gradiant mag in direction of gradiant. (local max)

$$\nabla(I * G) = (I_x, I_y)$$

$$\theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$



$$\theta^* = \text{round}\left(\frac{\theta}{45}\right) * 45$$

$0, 45, 90, 135 \ldots$

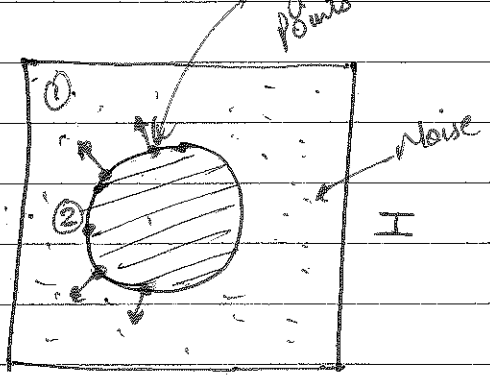
$$E(i,j) = \begin{cases} 1 & \text{if } \nabla(I * G) \text{ is a local max along } \theta \\ 0 & \text{otherwise} \end{cases}$$

2) hysteresis thresholding (tracking)

use τ_H to start tracking and τ_L to continue ($\tau_H > \tau_L$)

(a) Initialize array of visited pixels $V(i,j) = 0$

(b) Scan Image T-B, L-R:
if $!V(i,j)$ && $|\nabla I| > \tau_H$, start tracking on edge.



(c) Search for additional neighbors in direction orthogonal to ∇I such that $|\nabla I| > \tau_L$

* Edge detection performance evaluation:

1) Theoretical

2) Experimental (True positive, false positive)

3) Application outcome.

* Summary of (1.1.) Edge detection

Edge detection requires finding discrete image derivatives:

- forward difference
- Central difference
- Sobel
- Gaussian derivatives

} First order derivatives

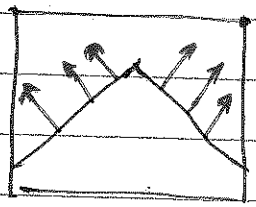
- Zero crossing of LOG (2nd order)
- Canny (directional derivative)

1. Feature Detection:

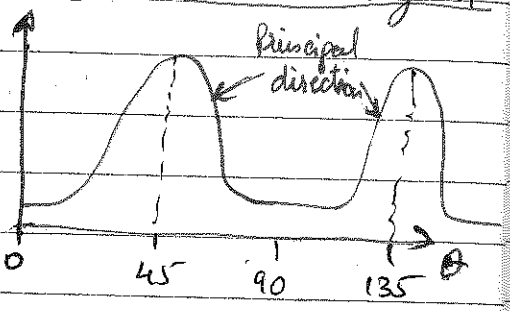
1.2 CORNERS

Corner = more than one direction in orientation histogram.

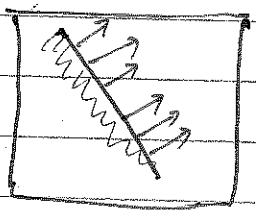
Corner

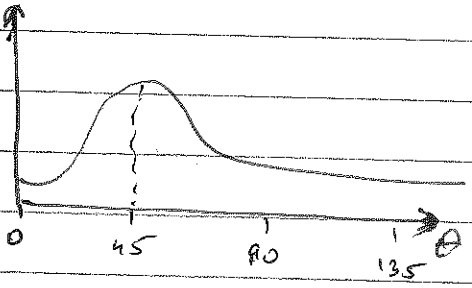


orientation histogram



edge





A corner is better defined than edge, it has a location. (better localization)

* Algorithm outline:

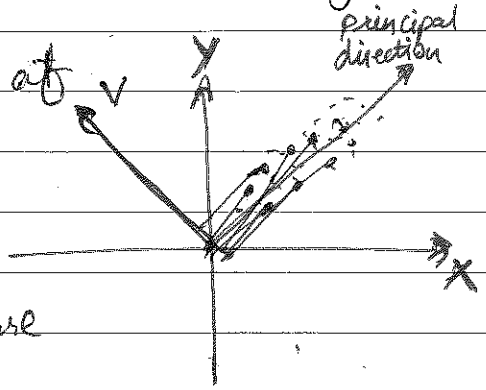
① Find correlation matrix of gradients in local ~~matrix~~ window.

② Find eigen values at correlation matrix (2×2)

③ detect corner in window if eigenvalues are sufficiently large.

* Algorithm is based on principle component Analysis (PCA)

Goal: find direction v s.t. projection of $\{g_i\}$ onto v is minimized.



① Each $\{g_i\}$ has $\begin{pmatrix} x_i \\ y_i \end{pmatrix}$ and are a point in coordinate system.

② We try to find a v such that the points $\{g_i\}$ Once projected on v minimize the spread.

So, we use $E(v)$ where $E(v)$ tells us how good the direction is.

$$E(v) = \sum_i (g_i \cdot v)^2 = \sum_i (g_i^T v)(g_i^T v)$$

$(g_i \cdot v)$ gives projections of $\{g_i\}$ on v.

$$= \sum_i (v^T g_i)(g_i^T v) = \sum_i v^T g_i g_i^T v$$

$$E(v) = v^T \left(\sum_i g_i g_i^T \right) v = v^T C v$$

$1\times 2 \quad 2\times 2 \quad 2\times 1$

1×1

$\{g_i\}$ is a 2×1 vector & so C is 2×2 matrix.

C is called ~~covariance~~ correlation matrix.

$$C = \sum_i g_i g_i^T = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i g_i & \sum y_i^2 \end{bmatrix}$$

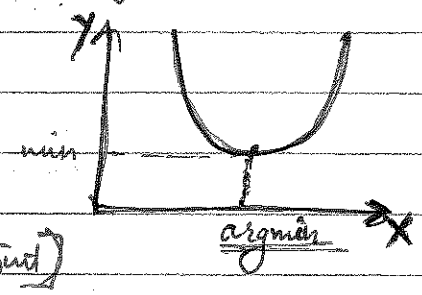
Note: In python we stack $\{g_i\}$ on top of each other to form a data matrix D & find correlation matrix.

$$D = \begin{bmatrix} g_1^T \\ g_2^T \\ \vdots \\ g_m^T \end{bmatrix} \quad \& \quad C = D^T D$$

Note ____ x ____

So,

$$\begin{cases} E(V) = V^T C V \\ V^* = \arg\min_V E(V) \end{cases} \qquad C = \sum_i g_i g_i^T$$



$\Rightarrow$

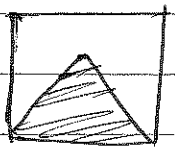
$\nabla E(V) = 0$ ← [is the minimum point]

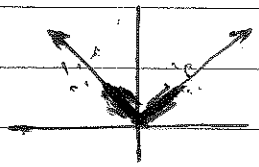
i.e. $2CV = 0 \Rightarrow$ solution is eigen vector belonging to smallest eigenvalues.

$$\underset{2\times 2}{\begin{bmatrix} C \end{bmatrix}} \begin{bmatrix} V \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

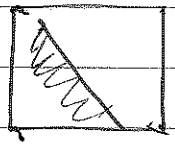
eigenvalue = variance in corresponding principal direction

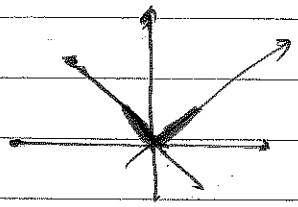
Lecture 9 - 0511

corner 

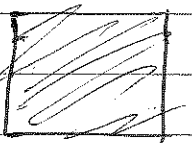


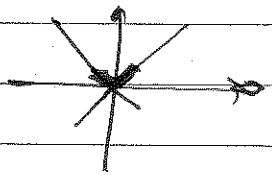
λ_1 large
λ_2 large

edge 



λ_1 small
λ_2 large

none 



λ_1 small
λ_2 small

Corner if $\lambda_1 \cdot \lambda_2 > \tau$

Corner detection Summary

(1) We scan the image T-B, L-R, at each pixel we select a local (neighborhood.

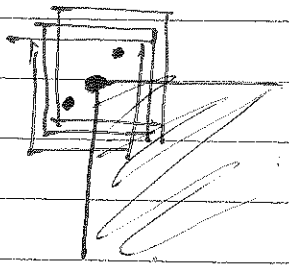
(2) Build correlation matrix $C = \sum_i g_i g_i^T$

(3) Compute eigenvalues of C

(4) Detect ~~eigen~~ corner if $\lambda_1 \cdot \lambda_2 > \tau$

* Non-maximum suppression

- If not done, each corner will lead to detecting multiple points.



So to get proper point we will need to:

(1) compute λ_1, λ_2 for all windows.

(2) select $\lambda_1 . \lambda_2 > \tau$ windows and sort in decending order.

(3) Select the top of the list as corner, and delete all other corners in its neighborhood.

(4) Stop once detecting X% of the points as corners.

This analysis needs to be done at multiple scales.

* Harris Corner detection :

(1) Compute correlation matrix C for window.

(2) Compute cornerness measure :

$$C_f(c) = \frac{\det(c)}{\lambda_1 - \lambda_2} - \frac{K\, tr^2(c)}{K(\lambda_1 + \lambda_2)^2}$$

(K) is selected By user

(3) detect corners where $C_f(c)$ is high

! trace is sum of diagonal elements

$K \in [0, 0.5]$ is a user parameter

if $K = 0$ $C_f(c)$ detects corners

if $K = 0.5$ $C_f(c)$ detects edges.

So,

$$C_f(c) = \det(c) - K\, tr^2(c)$$

$$= \lambda_1 \lambda_2 - K(\lambda_1 + \lambda_2)^2$$

$$= \underbrace{(1-2K)\lambda_1 \lambda_2}_{=0 \text{ if } K \text{ is } 0.5} - \underbrace{K(\lambda_1^2 + \lambda_2^2)}_{=0 \text{ if } K \text{ is } 0}$$

Corner detection edge detection

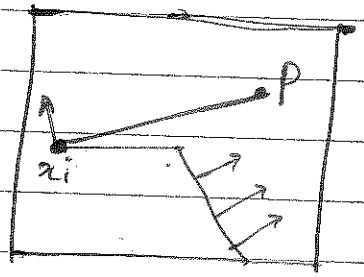
Alternative harris:

$$C_1(C) = \det(C) + K\left(\frac{tr(C)}{2}\right)^2$$

* Corner localization :

Given that there is a corner in a window find its location.

To determine if p is the corner connect each point x_i to p and project the gradiant at x_i onto $(x_i - p)$



The "best" p will minimize the sum of all projection

$$E(p) = \sum_i \left(\nabla I(x_i) \cdot (x_i - p)\right)^2 \quad \ldots [0 \text{ if corner}]$$

$$= \sum_i (x_i - p)^T \nabla I(x_i) \nabla I(x_i)^T (x_i - p)$$

$$= \sum_i (x_i - p)^T \left(\nabla I(x_i) \nabla I(x_i)^T\right) (x_i - p)$$

$1 \times 2 \quad (2 \times 1) \quad (1 \times 2) \quad 2 \times 1$

2×2

$$\begin{cases} E(p) = \sum_i (x_i - p)^T \left(\nabla I(x_i) \nabla I(x_i)^T \right) (x_i - p) \\ p^* = \underset{p}{\operatorname{argmin}} E(p) \end{cases}$$

$$\Rightarrow \nabla E(p) = 0$$

$$-2 \sum_i \left(\nabla I(x_i) \nabla I(x_i)^T \right) (x_i - p) = 0$$

$$\underbrace{\sum_i \nabla I(x_i) \nabla I(x_i)^T}_{C\ 2\times 2} \underbrace{p}_{2\times 1} = \underbrace{\sum_i \nabla I(x_i) \nabla I(x_i)^T x_i}_{2\times 1}$$

To solve it, isolate p on one side & move everything on other side.

$$C p^* = \sum_i \nabla I(x_i) \nabla I(x_i)^T x_i$$

multiply by C^{-1}

$$p^* = \left(\sum_i \nabla I(x_i) \nabla I(x_i)^T \right)^{-1} \sum_i \nabla I(x_i) \nabla I(x_i)^T x_i$$

$$p^* = C^{-1} \sum_i \nabla I(x_i) \nabla I(x_i)^T x_i$$

p^*: location of corner

C^{-1}: correlation matrix.

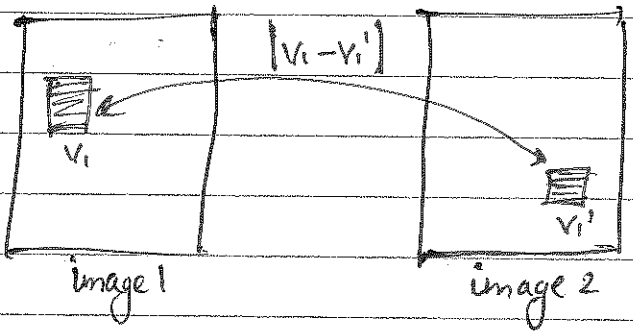
* Feature point characterization

Given a corner we would like to compare it to corners in another image and to use it to characterize the image.

⇒ feature point characterization (find feature vector)

*Applications:

- matching
- tracking
- correspondence
- Recognition



* Example methods:

- HoG - SIFT - SURF	OpenCV	- Shape context - spin image

* Desired Properties

- translation invariance → local window
- rotation invariance → histograms
- scale invariance → pyramid
- illumination invariance → gradients of gradients

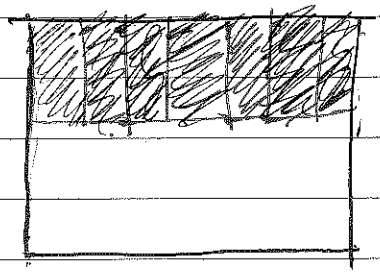
⇒ orientation histograms at different scales

+ rotation invariance
Rotate coord system (rotate hist) so that dominant direction at start.

+ Scale Invariance
Need representation at multiple scales.

* HOG method: (openCV)

(1) split each patch into cells (possibly overlapping)



(2) create orientation histogram in each cell (using edge or gradient directions, possibly weighted by distance from center of gradient magnitude)

(3) Concatenate orientation histograms

* SIFT (Scale Invariant Feature Transform):

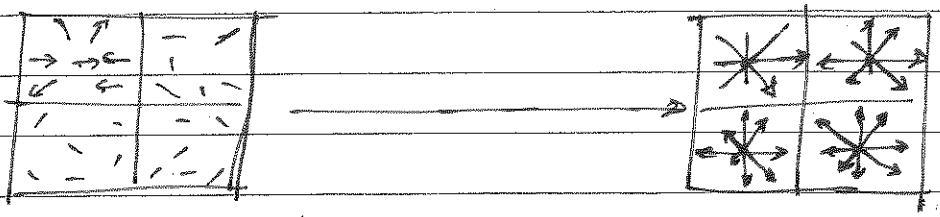


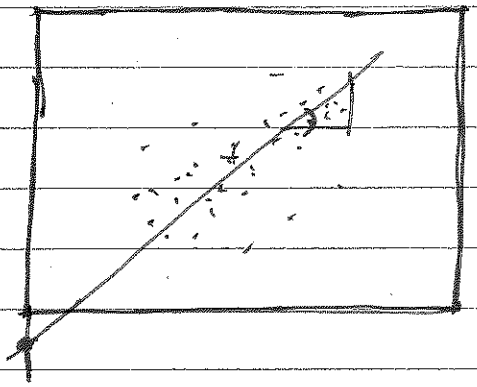
Image gradients → Keypoint descriptor

* - use weighted sum to create orientation histograms in cells, then concatenate
- Align histogram based on dominant direction (rotation invariance).

Lecture 10 - 05.12

MODEL FITTING

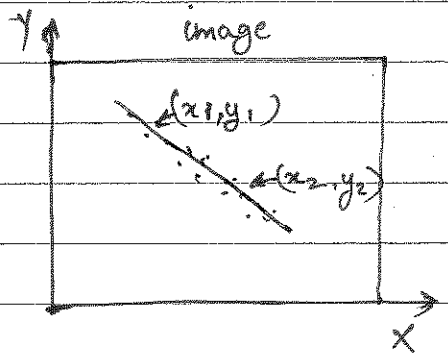
- Line ~~detection~~ Fitting

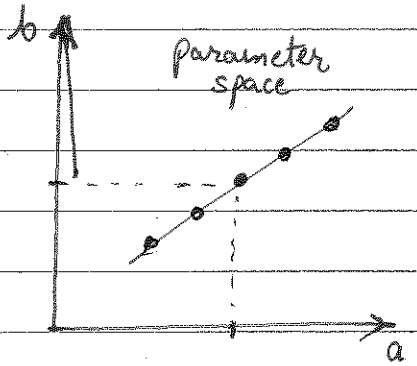


Tasks:
1) Grouping
2) Fitting

Two simultaneous tasks. Solving one makes the other easier.

* Hough transform:





$$y = ax + b$$

* multiple points on a line in the image correspond to a single point in parameter space (the parameters of the line).

* Detect lines by casting votes in parameter space.

so, $y = ax + b \longrightarrow b = y - ax$

given (x, y) scan a and compute b. Cast votes at locations (a, b) - a line.

- Parameters with highest votes wins in parameter space. Intersections in parameter space gives parameters common to multiple points in the image space.

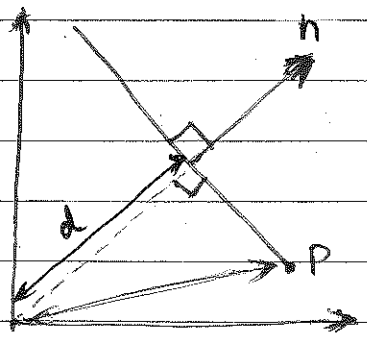
- When doing this on image votes need to be stored in a matrix. What so should be the intercepts of parameter space i.e slope & y-intercept.

slope can go to ∞ so it can be problemetic so that cannot be a good model.

→ Better line representation:

* Problem with $y = ax + b$ model:
  - range of a?
  - How to represent verticle lines?

Solution:



$n = (n_x, n_y)$ normal (orientation)

d = distance from origin

for point $p = (x, y)$ to be on the line:

$$(n_x, n_y) \cdot (x, y) = d$$ ← Line equation using n, d

$$\hat{n} \cdot p = d$$

where $\|(n_x, n_y)\| = 1$

* This is called as **Implicit** line equation:

$n = (n_x, n_y)$ where $\begin{cases} n_x = \cos\theta \\ n_y = \sin\theta \end{cases}$ angle of normal

equation is,

$$(n_x, n_y) \cdot (x, y) = d$$

$$\Rightarrow (\cos\theta)x + (\sin\theta)y - d = 0$$

$$(Ax + By + C = 0)$$

A, B: normal; C: negative dist from origin

* line parameters: θ, d

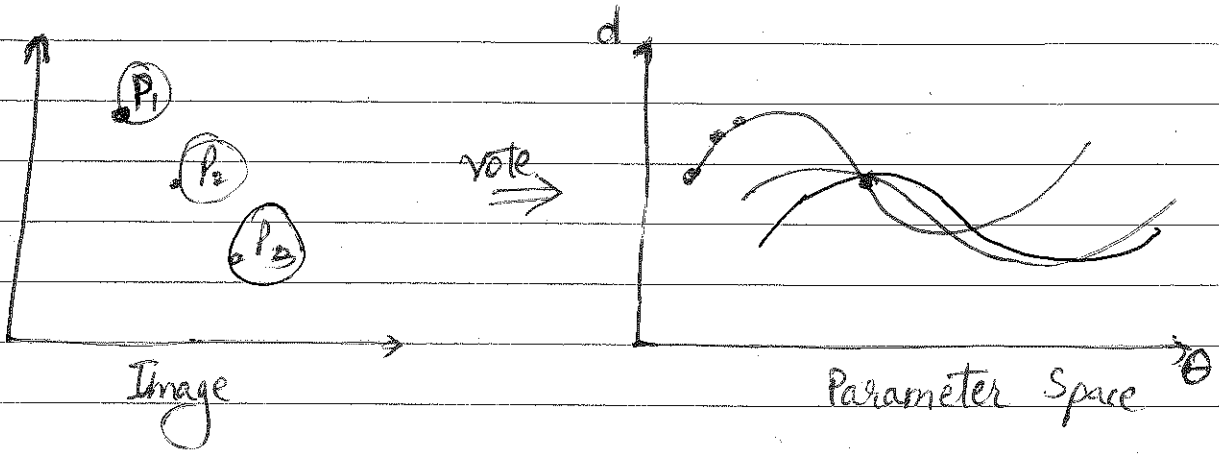
* Hough with explicit line equation:

→ ~~Explicit~~ Implicit line equation:

$$x\cos\theta + y\sin\theta = d$$

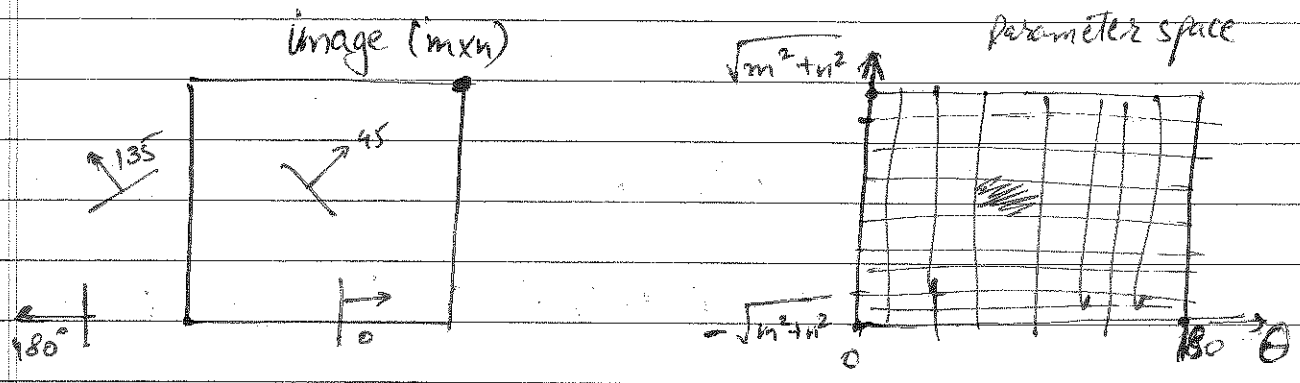
→ Vote by point (x, y):

for each θ_i vote $d_i = x\cos\theta_i + y\sin\theta_i$



* Practical issues

1) Parameter space size



2) Bin size : larger bins are more efficient but provide less localization. Small bins are more accurate.

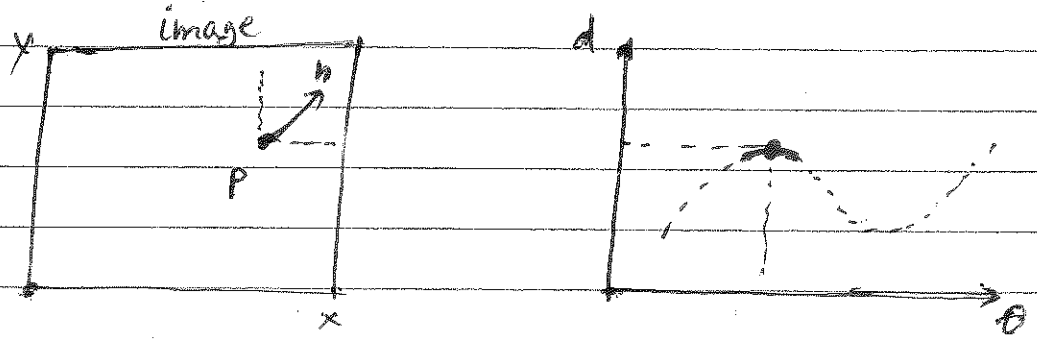
3) Peak detection : threshold & suppress close peaks.

* Hough with edge elements

Estimate a normal vector at each pixel

⇒

Each pixels provide a vote which is a small curve section (accounting for inaccurate normal direction)



A point (x,y) with normal θ votes for

$$d = x\cos(\theta + \alpha\Delta\theta) + y\sin(\theta + \alpha\Delta\theta)$$

where $\alpha \in [-1, 1]$

* Generalized Hough Transform

→ Objective : $f(x, y, a_1, \ldots, a_m) = 0$

eg: $f(x, y, \theta, d) = x\cos\theta + y\sin\theta - d = 0$

→ for each (x, y) scan $a_1, \ldots, a_{m-1}$ and vote for a_m

→ Problems :
- Sparse space with increased dimensions
- voting inefficient with increased dimensions.

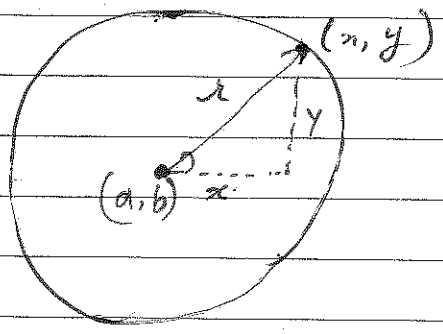
Example for a circle :

→ equation : $(a - x_1)^2 + (b - y)^2 = r^2$

or

$x = a + r\cos\theta$

$y = b + r\sin\theta$

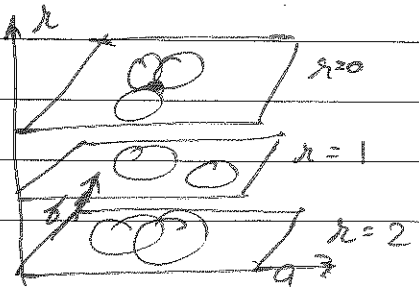


→ vote :

given (x, y) vote in each r plane using :

$a = x - r\cos\theta$ where $\theta \in [0, 360]$

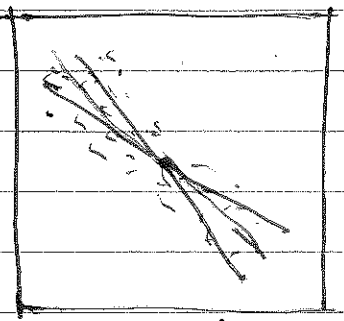
$b = y - r\sin\theta$



~~Hough over here~~

* Model Fitting

After hough model



Given data $\{(x_i, y_i)\}_{i=1}^{m}$

& model $M_p(x_i)$:

- ~~determine~~ define objective: $E(p) = \sum_i (M_p(x_i) - y_i)^2$

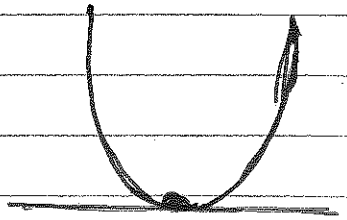
- minimize objective: $p^* = \underset{p}{\operatorname{argmin}}\ E(p)$

Line fitting: given a group of points (eg. neighboring a line detected using Hough) find a more accurate model.

given $\{(x_i, y_i)\}_{i=1}^{n}$ find the parameters of line fn

$y = ax + b$

$$\begin{cases} E(a,b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2 \\ a^*, b^* = \underset{a,b}{\operatorname{argmin}}\ E(a,b) \end{cases}$$



$\Rightarrow \nabla E(a,b) = 0 \Rightarrow a^*, b^*$

Lecture - 11 Line fitting

* Given data $\{(x_i, y_i)\}_{i=1}^{m}$ and model $M_p(x_i)$:

- define objective : $E(p) = \sum_i (M_p(x_i) - y_i)^2$

Sum of difference b/w prediction.

- minimize objective : $p^* = \underset{p}{\operatorname{argmin}}\, E(p)$

Line fitting :

$$E(a,b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2$$

$$\nabla E(a,b) = \begin{bmatrix} \frac{\partial E}{\partial a} \\ \frac{\partial E}{\partial b} \end{bmatrix} = \begin{bmatrix} \sum 2 (y_i - ax_i - b)(-x_i) \\ \sum 2 (y_i - ax_i - b)(-1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \sum 2x_i^2 & \sum 2x_i \\ \sum 2x_i & \sum 2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum 2x_i y_i \\ \sum 2y_i \end{bmatrix}$$

A $\quad X$ $\quad B$

Logic: $(Ax = b) \times A^{-1}$

$$A^{-1}Ax = A^{-1}B$$

$$x = A^{-1}B$$

$\Rightarrow x = A^{-1}b$ where

$$A = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i \\ \Sigma x_i & m \end{bmatrix} \quad x = \begin{bmatrix} a \\ b \end{bmatrix} \quad b = \begin{bmatrix} \Sigma x_i y_i \\ \Sigma y_i \end{bmatrix}$$

while doing this in python use =solve(A, B) instead of =inv(A).B

Poor model –

– the solution obtained is optimal in the sense of minimizing the objective:

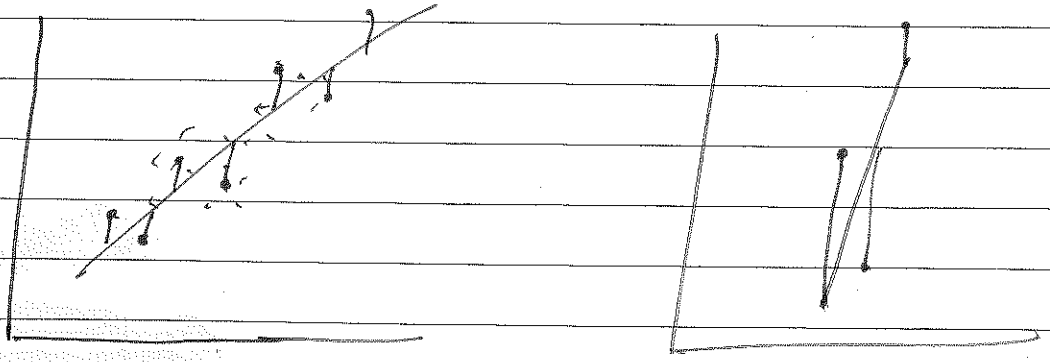
$$E(a,b) = \sum_i \underbrace{\left(y - (ax_i + b)\right)}_{e_i}^2$$

y_i

e_i ← algebraic distance

– The problem is objective itself.

– This take vertical error, which is bad.



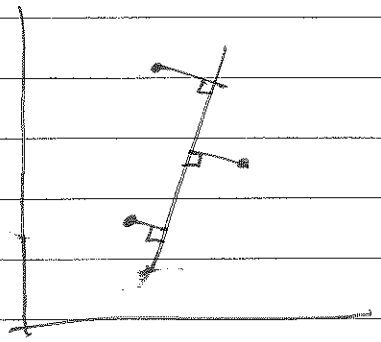
– So this becomes inaccurate.

Solution: To use ⊥ projection to a line i.e geometric distance.

Minimize geometric distance:

- line in homogeneous coordinates

- point P on the line l if:



$l^T p = 0$

$(a,b,c)^T$ $(x,y,1)$

$l^T p = 0 \Rightarrow ax + by + c = 0$

$ax + by + c = 0$

normal vector

-ve dist from origin

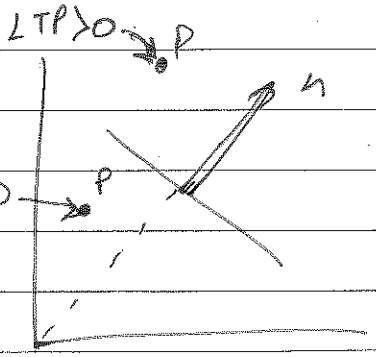
for p to be on the line:

$p \cdot n = d$

$n_x X + n_y Y - d = 0$

* if $n = (a,b)$ is not normalized, c is only proportional to distance.

when p is not on the line l, $l^T p$ is the geometric distance of p from l.



- $l^T p$ is also the algebraic distance (deviation from the expected value of 0)

- In line fitting the algebraic error has a meaning of being a geometric error.

* Geometric Line fitting :

$$\begin{cases} E(l) = \sum_{i}^{m} (l^T p_i)^2 & p_i = (x_i, y_i, 1) \\ l^* = \underset{l}{\operatorname{argmin}}\ E(l) \end{cases}$$

3×3 → $[3\times1 \mid 1\times3]$

$$E(l) = \Sigma\, l^T p_i p_i^T l = l^T \underbrace{\Sigma p_i p_i^T}_{S} l = \underset{1\times3}{l^T} \underset{3\times3}{S} \underset{3\times1}{l}$$

$$\nabla E(l) = 0 \Rightarrow Sl = 0$$

$\Rightarrow l$ = eigenvalues of S belonging to zero eigenvalue

↓

[eigen value belonging to smallest eigen value.]

to evaluate solution compute $\left(\frac{E(l)}{m}\right)$ error/point in pixels

* correlation metrix (structure tensor)

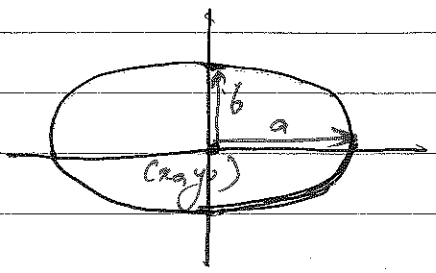
$S = \Sigma\, p_i p_i^T$ = correlation matrix of points $p_i(x_i, y_i)$

$$S = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i y_i & \Sigma x_i \\ \Sigma x_i y_i & \Sigma y_i^2 & \Sigma y_i \\ \Sigma x_i & \Sigma y_i & m \\ & & \Sigma 1 \end{bmatrix} = D^T D \qquad D = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 2 \\ \vdots & & \\ x_m & y_m & 1 \end{bmatrix}$$

Ellipse fitting

Explicit axis-aligned equation:

$$\left(\frac{x-x_0}{a}\right)^2 + \left(\frac{y-y_0}{b}\right)^2 = 1$$

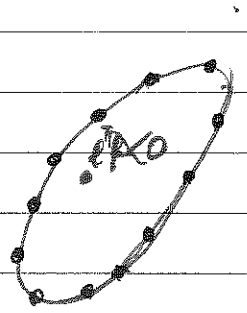


Circle equation is

$$(x-x_0)^2 + (y-y_0)^2 = r^2 \longrightarrow \left(\frac{x-x_0}{r}\right)^2 + \left(\frac{y-y_0}{r}\right)^2 = 1$$

* Implicit equation (Conic curve)

$$\begin{cases} ax^2 + bxy + cy^2 + dx + ey + f = 0 \\ b^2 - 4ac < 0 \end{cases}$$



$$l^T P = 0$$

(a,b,c,d,e,f) ← $(x^2, xy, y^2, x, y, 1)$

- convert input points

$$\{(x_i, y_i)\} \longrightarrow \{P_i\} \qquad P_i = (x^2, xy, y^2, x, y, 1)$$

* Find parameters ℓ by minimizing algebraic distance:

$$E(\ell) = \sum_{i=1}^{m} (\ell^T P_i)^2 = \ell^T S \ell$$

$$S = \sum P_i P_i^T \leftarrow 6 \times 6$$

$$\ell^* = \underset{\ell}{\operatorname{argmin}}\ E(\ell) \quad \text{such t. } b^2 - 4ac < 0$$

* Rewriting the constant:

$$b^2 - 4ac < 0 \iff \ell^T C \ell = -1$$

$$C = \begin{bmatrix} 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* Modified objective:

$$E(\ell) = \ell^T S \ell + \lambda \underbrace{(\ell^T C \ell + 1)}_{0}$$

$$\ell^* = \underset{\ell}{\operatorname{argmin}}\ E(\ell)$$

λ — lagrange multiplier

$$\nabla E(\ell) = 0$$

$$\Rightarrow 2 S \ell = 2 \lambda C \ell$$

$$\Rightarrow \ell = \lambda S^{-1} C \ell \implies \ell^* \text{ is the eigenvector of } S^{-1}C$$

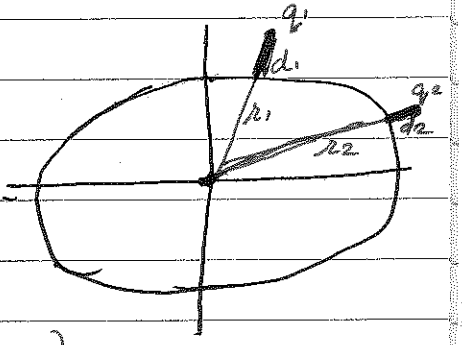
belonging to its −ve eigenvalue

* algebraic distance of ellipse:

ℓ : 6x1

When x_i is on the ellipse: $\ell^T x_i = 0$

- when x_i is off the ellipse: $q_i = \ell^T x_i$ provides a measure for distance from the ellipse.

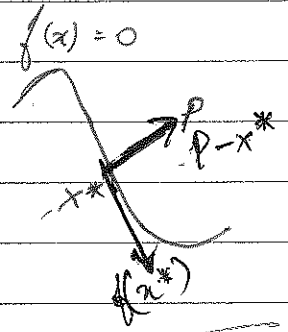


algeric distance $\equiv q_i \equiv \ell^T x_i \cong \frac{d_i}{d_i + r_i}$

$d_1 \cong d_2$ but $q_1 > q_2$ (because $r_2 > r_1$)

* Geometric distance of ellipse:

geometric distance from a point to arbitrary implicit curve: $f(x) = 0$



$$\begin{cases} d(p, f) = |p - x^*| \\ x^* \text{ is closet point} \end{cases}$$

* solution:

$$d(p, f) = |p - x^*| = \frac{|f(p)|}{|\nabla f(x^*)|}$$

algebraic distance

To find x^* solve:

$$\begin{cases} f(x^*) = 0 \\ (p - x^*) \cdot t(x^*) = 0 \end{cases}$$

tangent at x^*

* to find tangent at x^*

1) compute gradient : $\nabla f(x^*) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T$

2) Rotate by $-90°$: $t = R(-90)\,\nabla f(x^*) = \left[\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right]^T$

* Approximated solution:

$$d(p, f) = |p - x^*| = \frac{|f(p)|}{|\nabla f(x^*)|} \simeq \frac{|f(p)|}{|\nabla f(p)|}$$

$|p - x^*|$: geometric distance

$\frac{|f(p)|}{|\nabla f(p)|}$: algebraic distance

$\Rightarrow$ reduce the algebraic distance from points with large gradient $|\nabla f(p)|$

* Update geometric distance fitting:

$$E(l) = \sum_i \frac{|f(p_i, l)|}{|\nabla f(p_i, l)|} \leftarrow \sum \frac{(l^T p_i)^2}{\nabla (l^T p_i)^2}$$

where $f(p_i, l) = l^T p_i$

$l^* = \underset{l}{\text{argmin}}\, E(l)$ s.t. $b^2 - 4ac < 0$

No explicit solution to $\nabla E(\ell) = 0$

$\Rightarrow$ iterative numerical solution (eg: Gradiant descent).

Lecture 12

☆ Arbitrary model fitting (eg. splines)

- Given groupoints $\{p_i\}$ and a parametric model $M(P_i, l)$ with parameters l:
- Define distance between point P_i & model:
  $d_i = d(M(P_i, l), P_i)$
- Define objective:

$$E(l) = \sum d_i^2 = \sum d^2(M(P_i; l), P_i)$$

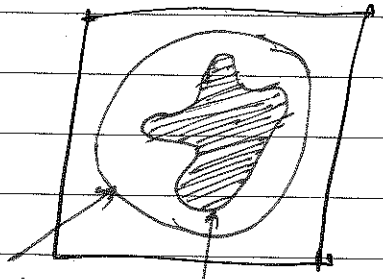
- Minimize objective to find parameters

$$l^* = \underset{l}{\operatorname{argmin}}\ E(l)$$

☆ Others models

* Active Contours (snakes)

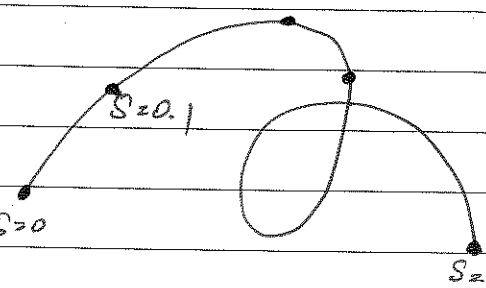
— Gradually deform initial contour to fit object boundries.
→ Iterations



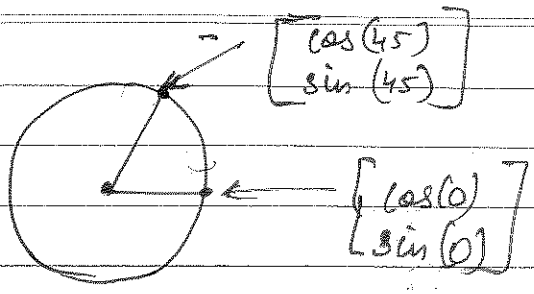
— Use parametric curves $\phi(s)$ to represent curves

Parametric curve

$$\phi(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$$



Eg: $\phi s = \begin{bmatrix} \cos(s) \\ \sin(s) \end{bmatrix}$



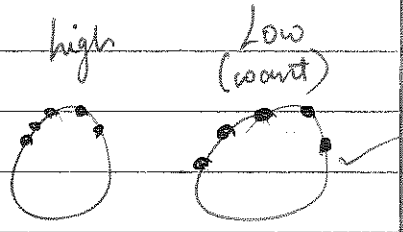
* error functional

The unknown we are ~~finding~~ seeking is a function $\phi(s) \Rightarrow$ error ~~measure~~ functional

$$E[\phi(s)] = \int_{\phi(s)} (\alpha(s) E_{cont} + \beta(s) E_{curv} + \gamma(s) E_{img}) ds$$

internal energy — external energy

$\alpha(s)$, $\beta(s)$, $\gamma(s)$ are coefficients of the different energy terms.



continuity energy : $E_{cont} = \left| \frac{d\phi}{ds} \right|^2$

curvature energy : $E_{curve} = \left| \frac{d^2\phi}{ds^2} \right|^2$

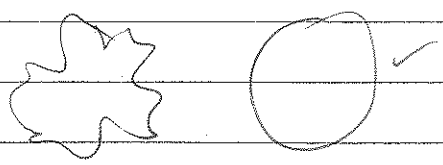
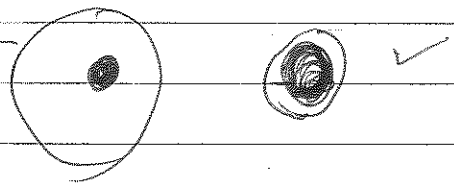


image energy : $E_{img} = -\left| \nabla I \right|^2$



* Discrete functional :-

Discrete case : $\phi(s) \longrightarrow \{P_i\}_{i=1}^{n}$

$$E_{cont} = \left|\frac{d\phi}{ds}\right|^2 = \left|P_{i+1} - P_i\right|^2$$

$$E_{curv} = \left|\frac{d^2\phi}{ds^2}\right|^2 = \left|(P_{i+1} - P_i) - (P_i - P_{i-1})\right|^2$$

$$= \left|P_{i+1} - 2P_i + P_{i-1}\right|^2$$

$$E_{img} = -\left|\nabla I\right|^2$$

coordinates of P_i ; unknown → model parameters ; Avg distance b/w points to prevent shrinking

$$E(\{P_i\}) = \sum_{i=1}^{n} \alpha_i \left(\left|P_{i+1} - P_i\right| - d\right)^2 + \sum_{i=1}^{n} \beta_i \left|P_{i+1} - 2P_i + P_{i-1}\right|^2 - \sum_{i=1}^{n} \gamma_i \left|\nabla I(P_i)\right|^2$$

$\alpha_i, \beta_i, \gamma_i$ are user selected parameters. $\{P_i\}$ are the unknown model parameters. d is a computed parameter.

(also zero β_i at corners to allow discontinuity)

↳ set β_i to zero if curvature @ P_i is $> \tau$

* To minimize the error functional E:

- start with initial guess $\{P_i\}_{i=1}^{n}$

- compute d coverage distance between points

- for each point P_i try to minimize E by testing what happens when moving to neighboring locations

- Repeat ~~if~~ while E is decreasing

* Selecting coefficient

- select α_i β_i γ_i to normalize the different terms to a similar scale
- To allow for piecewise curves, set $\beta_i = 0$ to points with high curvature:

  ie.

  when $|P_{i+1} - 2P_i + P_{i-1}| > \tau$

Robust Estimation

* Naive approach:

- Fit model to all points
- Compute distance at each point from model
- Discard points with largest distance
- Fit model to remaining points

problemies: initial model is inaccurate and so we drift in the wrong direction.

Approaches:
- M-estimates
- RANSAC

* M-estimators

- mean square error (mse) fitting:

$$E(\theta) = \sum d^2(x_i;\theta)$$ ← eg: $d^2 = (e^T x_i)^2$ for line fitting

- robust estimates:

$$E(\theta) = \sum \rho_a(d(x_i;\theta))$$

MSE is a special case where $\rho_a(x) = x^2$

— Geman - McClure estimator

Better than robust

$$\rho_\sigma(x) = \frac{x^2}{x^2 + \sigma^2}$$

$x \gg \sigma \Rightarrow \rho_\sigma(x) = 1$

$x \ll \sigma \Rightarrow \rho_\sigma(x) = \frac{x^2}{\sigma^2}$

larger σ

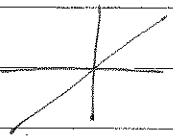
$\Downarrow$

wider valley

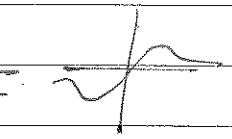
$$E(\theta) = \sum \rho_\sigma(d(x_i; \theta))$$

$$\nabla E(\theta) = \sum \frac{\partial}{\partial d} \cdot \rho_\sigma(d) \frac{\partial}{\partial \theta} d(\theta)$$

for, $\rho_\sigma(d) = d^2 \Rightarrow \frac{\partial \rho}{\partial d} = 2d$



$$\rho_\sigma = \frac{d^2}{d^2 + \sigma^2} \Rightarrow \frac{\partial \rho}{\partial d} = \frac{2d\sigma^2}{(d^2 + \sigma^2)^2}$$



* Selecting bandwidth parameter

— large $\sigma \Rightarrow$ include more points

— small $\sigma \Rightarrow$ include fewer points

— variable estimation:
start with large σ and decrease as converging.

$$\sigma^{(n)} = 1.5 \text{ median } \{ d(x_i ; \theta^{(n-1)}) \}$$

$\sigma^{(n)}$: estimate at step n

$\theta^{(n-1)}$: parameters at step n-1

M-estimator summary:

1) Draw a large set of points uniformly at random

2) select initial value of σ

3) Fit model $\rightarrow \theta^{(i)}$

4) compute $\sigma^{(i)}$ using median distance of points

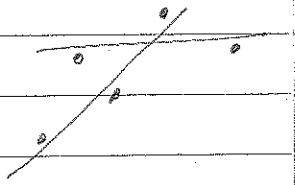
5) continued while objective is decreasing

* To overcome incorrect initial guess of σ repeat several times and select best solution (smaller objective)

* RANSAC:

Random Sample Consensus (RANSAC):

- perform multiple experiments
- choose best results
- use small sets in hope that at least one set will not have outliers.



Parameters:

n = # points drawn at each evaluation
d = min # points needed to estimate model
K = # trials
t = distance threshold to identify inliers

* Repeat K times:

- Draw n points uniformly at random (with replacement)
- fit a model to points
  - Find inliers in entire set [distance < t]
  - Recompute model (if at least d inliers]
  - update parameters (K, t)

* Choose best solution:

→ Largest consensus set
- (or smallest error)

Lecture 13 - 07/21

(Model fitting (contd) - - - -)

* How to identify threshold (t) :

to estimate t use median distance from model.

$$t = 1.5 \times \text{median}$$

* To estimate (K) use :

(user selected) p : with probability of p at least one experiment does not have outliers (eg. p = 0.99)

(estimated) w : probability that a point is an inlier (initially ½) $w = 0.5$

So probability that all k experiments failed :

$$(1-p) = (1-w^n)^k$$

$$\log(1-p) = k \log(1-w^n)$$

$$k = \frac{\log(1-p)}{\log(1-w^n)}$$

large p ⟶ large k
small w ⟶ large k

$w \leftarrow \frac{\#\text{ inliers}}{\#\text{ points}}$

update w, k every interation but set upper bound for K.

Segmentation and Classification

Segmentation:

2class segmentation – foreground & Bg

3class segmentation – detect vegetables from 3 colors.

It is difficult to do with just colors.

Semantic segmentation – most difficult. – identify everything as class.

Perceptual segmentation problem (more difficult):
Computer cannot perceptualize objects.

* Problem Statement:

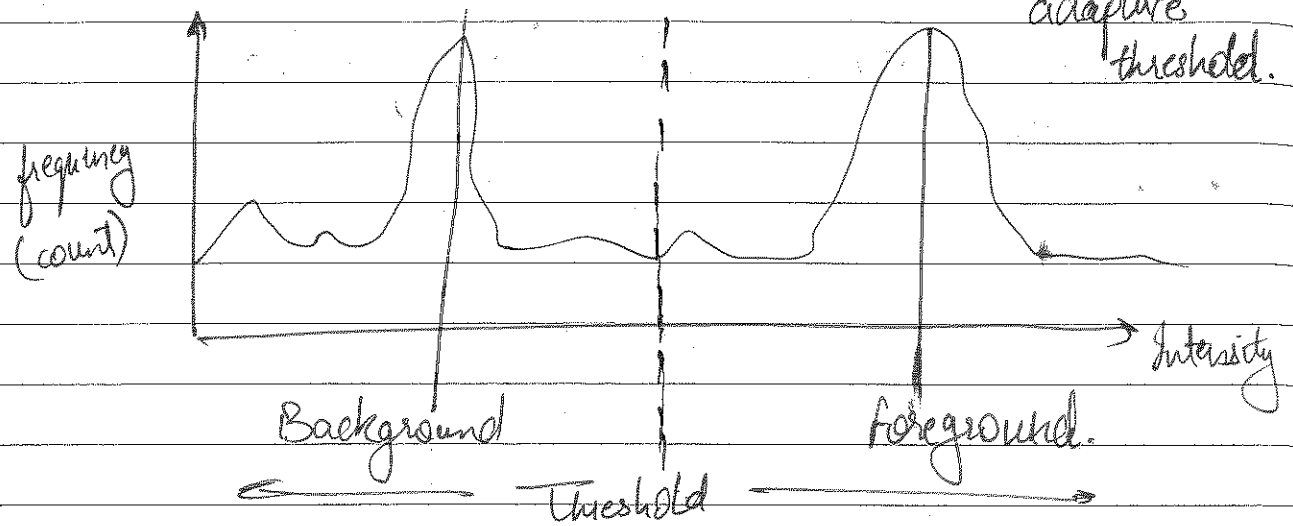
- Separate objects from background.
- Find contours of object
- Semantic image segmentation: label each pixel in the image with class label.

* Color segmentation

- Use color thresholds to indentify objects

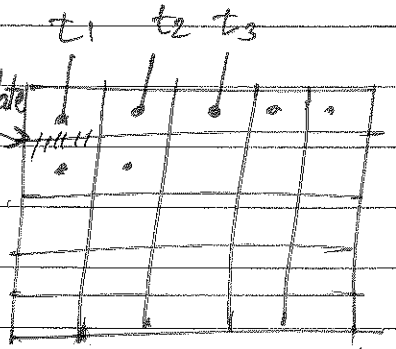
Problem: how to identify colors threshold.

Solution: Use Intensity histogram.



* But this can still skip certain objects,

Solution : use cells



- Break image into cells
- find threshold in each block
- Interpolate threshold values to create segmentation surface.
- at each pixel location compare image intensity to segmentation surface.

So this is adaptive threshold.

* **Spacial context :** to group pixels together which have similar characteristics. This gives super pixels.

Agglomirative segmentation : Super clusters

- Start with each pixel in a separate clusters
- Merge clusters with small distance, color & edge difference
- Repeat while clusters are not ~~satisfied~~ satisfactory

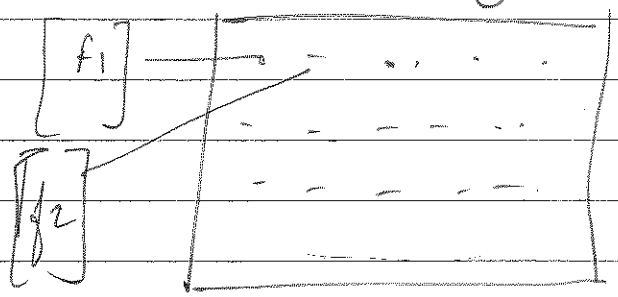
Divisive segmentation:
- start with all pixels in one cluster
- split clusters to provide large distance between them.
- Repeat while clusters are not satisfactory.

* Feature based segmentation

• Define feature vector at each pixel x:

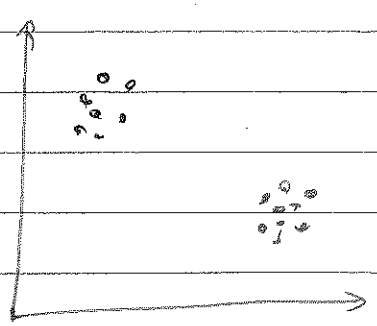
$$f(x) = \begin{bmatrix} x \\ I(x) \\ L(x) \end{bmatrix} \begin{array}{l} \leftarrow \text{location} \\ \leftarrow \text{intensity} \\ \leftarrow \text{local characteristic (eg texture)} \end{array}$$

• Apply clustering



→ Clustering (Machine Learning Concept)

- Each pixel is assigned a feature vector
- To define a cluster group pixels with similar feature vectors.
- The similarity in each cluster should be high compared to the similarity between clusters.



* fundamental problem in clustering:

- Need to compute cluster parameters and assign items ~~and~~ to clusters
- Joint estimates is difficult.

* Issues:
- what is the number of clusters?
- How to compute similarity within clusters and distance between clusters?
- what features to use?

* K means (clustering algo):

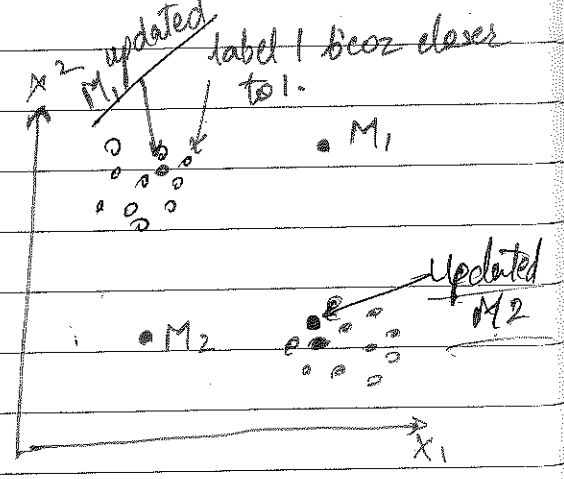
- select K
- select initial guess of means: $m_1, \ldots, m_k$
- Repeat while m_j change:

① $l_i = \underset{j \in [1,K]}{\operatorname{argmin}} ||f_i - m_j||^2$

for each piece (assign labels)

② $\delta_j = \{ i \mid l_i = j \}$

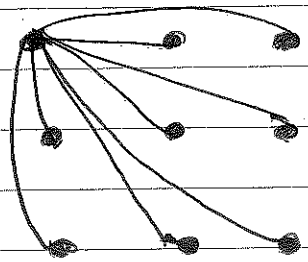
③ $m_j = \frac{\sum_{i \in \delta_j} f_i}{\# \delta_j}$ ← average new center (recompute mean)



* simultaneous solution of the two problems:
  - find cluster centers
  - assign points to clusters

hold one parameter fixed and change other to minimize error.
⇒ coordinate descent on $E(\{l_i\}, \{m_j\}) = \sum \|f_i - m_{l_i}\|^2$

* Graph cuts (other method than clustering):
to identify clusters / super pixels.

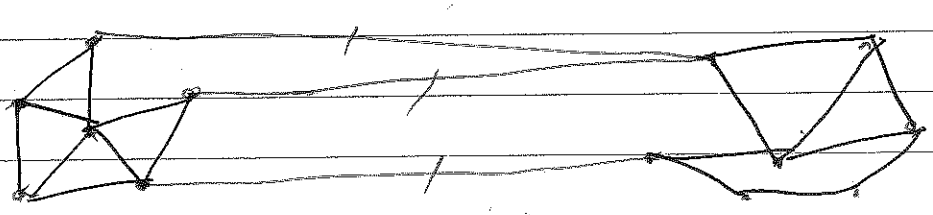


- Represents image using graph.
- each pixel is connected to all other pixels.
- The weight on the link between nodes p & q represents the similarity b/w them (intensity + location).

$$w_{pq} = \exp(-\|f(p) - f(q)\|)$$

if $f(p)$ similar to $f(q)$ then $w_{pq} = 1$
else $w_{pq} = 0$

- Delete links to create segments.
- Remove low similarity links.
- Nodes in each cluster should have strong links.



* possible graph cuts remove links to create disconnected subgraphs.
  - The cost of a cut is the sum links being removed:

$$cut(A,B) = \sum_{p \in A, q \in B} w_{p,q}$$

  - Minimum cut: cut with lowest cost.

* But this is not desirable as each pixel will end up being 1 subgraph.
  - a normalized cut assigns cost taking into account the size of produced clusters.

$$Ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

$$Vol(A) = \sum_{p \in A, p \in A \cup B} w_{p,q}$$

* finding normalized cuts:
  - the weighted degree d_i of a node is the sum of all edges connected to it:

$$d_i = \sum_{j=1}^{n} w_{ij}$$ ← Similarity

W = Similarity matrix

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nn} \end{bmatrix}$$

weighted degree of each node → $D = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix}$

- Defining a cut:

$$X = \{1, -1\}^n \qquad X(i) = 1 \iff i \in A$$
$$X(i) = -1 \iff i \in B$$

$$Ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

$$= \frac{\sum_{\substack{x_i>0 \\ x_j<0}} (-w_{ij} x_i x_j)}{\sum_{x_i>0} d_i} + \frac{\sum_{\substack{x_i>0 \\ x_j<0}} (-w_{ij} x_i x_j)}{\sum_{x_j<0} d_j}$$

* Define:

$$k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i} \qquad b = \frac{k}{1-k} = \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}$$

$$y = (1+x) - b(1-x)$$

Convert the problem:

$$\begin{cases} \min_x Ncut(x) \overset{y}{=} \min_y \dfrac{y^T(D-W)y}{y^T D y} \\ \text{s.t} \quad y^T D 1 = 0 \\ 1 = [1 \dots 1] \end{cases}$$

* since its hard to solve with discrete as $[1, -1]$ needs to be assigned, we make it continuous domain

solve: $\min_y \frac{y^T(D-w)y}{y^T D y}$ s.t. $y^T D 1 = 0$

↑ Rayleigh quotient

$\Rightarrow (D-w)y = \lambda D y$

- the first eigen vector (belonging to $\lambda = 0$) is $\underline{1} = [1 \ldots\ldots 1]$

- the second smallest eigenvector is the solution. (the one that is not zero)

* Spectral Clustering (like graphcut somewhat):

- has a degree matrix as before.
- has an adjacency matrix (W) which is very similar but just that the similar matrix have a value of 1 else a 0.

- graph laplacian: $L = D - W$
- L is symmetric and positive semidefinite
- the sum of every row & column in L is ≥ 0.
- the e.vector belonging to second eigen value as smallest value (0) has a vector $[1 \; \ldots \; 1]$
- For every $x \in \mathbb{R}^n$: $x^T L x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} |x_i - x_j|^2$

* so to compute spectral cluster:

- compute eigen vector of L
- Stack K smallest eigen vectors: $U = [u_1 | \; | \; \ldots \; | u_K]$ ($n \times K$; K dimension) with non zero eigen values.
- define y_i as the i-th row of U (ie map data to K-dimensional space) $nD \rightarrow KD$
- cluster the n rows y_i with K-means (i.e each cluster in K-D space instead of n-D)

* Classification:

- Classification
- Classification + localization
- object detection
- Instance segmentation
- semantic segmentation

* Problem with object recognition?

- Semantic gap (pixels vs labels):
  - pixels ~~valut~~ values as descriptors are sensitive to small variations.
- object ~~rego~~ recognition needs to be invariant
  - pose
  - illumination
  - deformation (articulated objects)
  - occlusion
  - background
  - natural variability (eg. cars)

* Want:

- generalization (invariance)
- extend to other problems

* Need:
  - feature extraction (invariance)
  - classification algorithm.

⇒ Some traditional object recognition methods:-

today state of the art is to use deep learning

Eigen faces:

- map image (eg. 100x100) to lower dimensional vectors (eg 64) using PCA.
- ~~massive~~ measure similarity to templates in lower dimensional space (loss sensitive to small variations.)

Bag-of-words

- extract features from image (eg SIFT or HOG)
- cluster features to create a codebook (dictionaries)
- compute a distribution of code words in each class
- classify using distribution of code words.

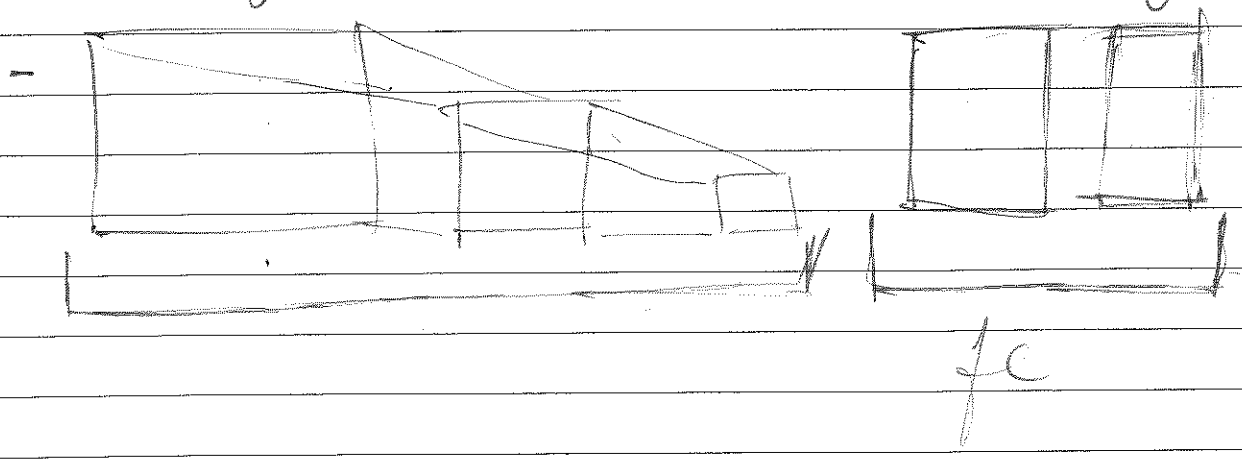
*Convolutional neural networks (CNN):

codebook — Convolutional neural networks (CNN) learn feature extraction and classification
- learn from examples instead of coding algorithms
- Architecture for one class can work for others by changing data → class generalization.

Layers of convolution | classification layers

learned feature extraction

- Combine features at multiple scales (multiple scale analysis)

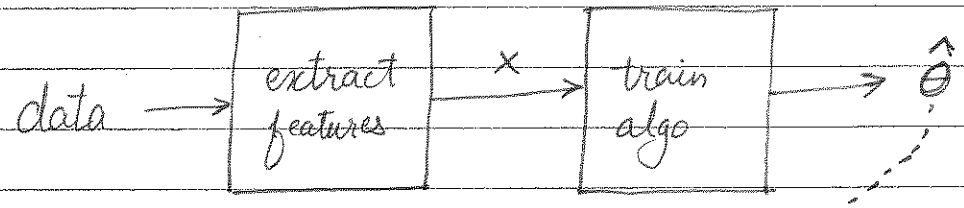


Lecture 14

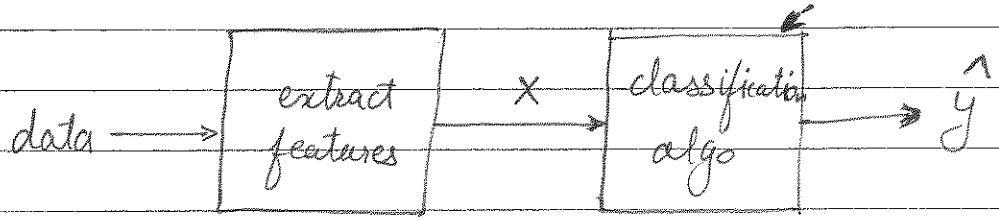
☆ DEEP LEARNING (DATA-DRIVEN

* Training & inference :-

Training :

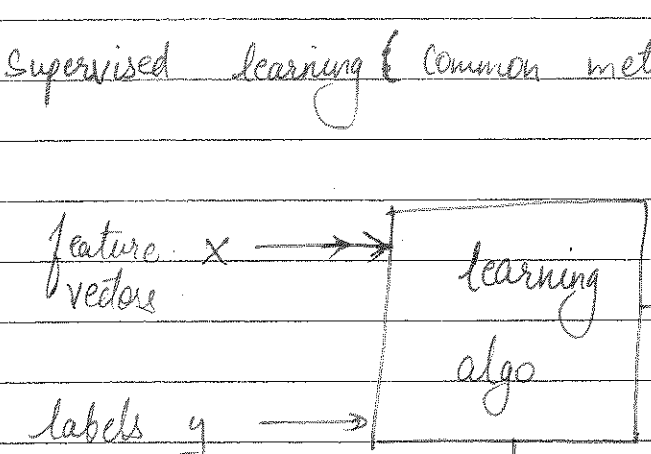


Inference :



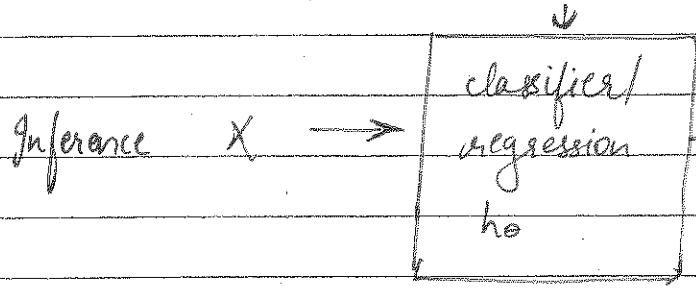
* Supervised learning (common method) :

$\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$ $x^{(i)} \in \mathbb{R}^n$



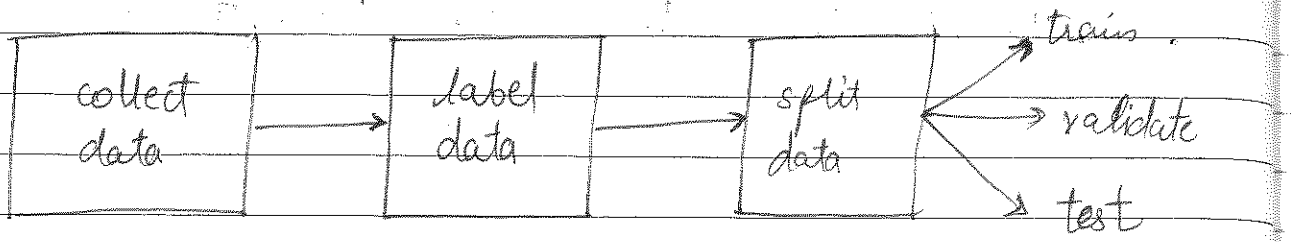
$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$

loss function compares $y, \hat{y}$



discrete or continuous output

* Data collection :



- Train/test on different collection (eg. K fold cross validation for small data available)
- Performance based on testing error (generalization vs. overfitting)
- Large models have more capacity but require more data (exponentially) and many overfit.
- Parameters :
  - Learn model parameter (training)
  - Hyper-parameters are specified parameters.

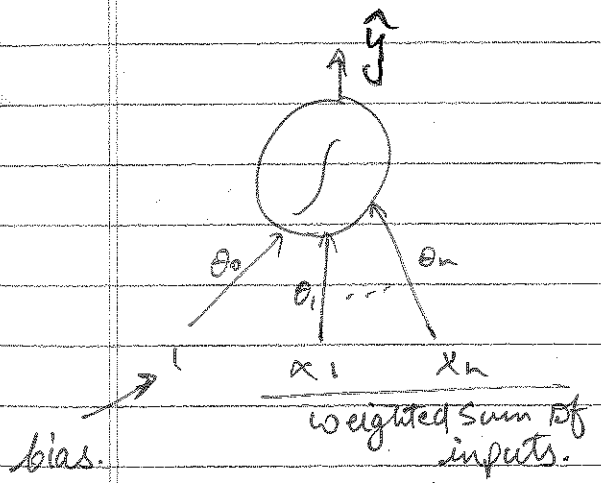
Deep Learning

→ Alexnet
→ ResNet
→ Google Net / inception
→ Yolo - real tim
→ U-net (image segmentation)
→ Deeplab (semantic segmentation)
→ triple net
→ Deep Pose
→ VGG (classification - feature extraction)
→ Cycle GAN (Image mapping)

→ Deep dream (Image hallucination)
→ Image enhancement (Chen et.al 2018)

★ Neural Networks

- A neural network is a collection of artificial neurons.
- artificial neurons are linear classifiers with activation



$$\hat{y} = f(\theta x_1 + \dots + \theta_n x_n + \theta_0) \quad \leftarrow \text{activation function}$$

$$= f(\theta^T x + \theta_0) \quad \theta_0 : \text{bias / Threshold}$$

(STEP) simple activation func:

$$= \begin{cases} 1 & \text{if } \theta^T x > -\theta_0 \\ 0 & \text{otherwise} \end{cases}$$

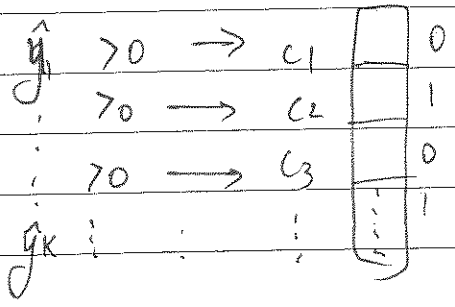
- fire if weighted sum of inputs is ~~greater~~ above threshold/bias.

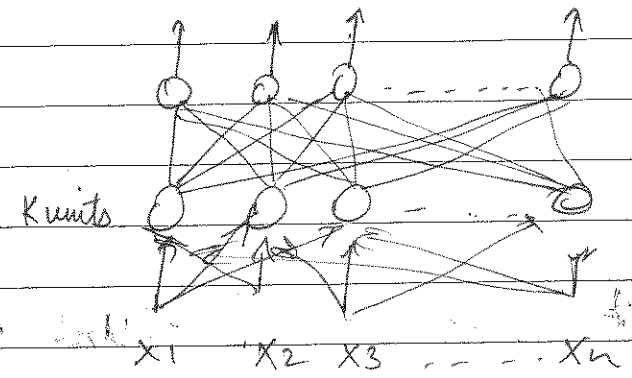
→ $\theta_0\ \theta_1\ \theta_2 \dots \theta_n$ are learnt during process

→ They are called linear classifiers, although activation function may not be ~~linear~~ linears.

$$\underbrace{\hat{y}}_{K\times 1} = f(x;\theta) = \underbrace{\theta^T}_{K\times n} \underbrace{x}_{n\times 1} + \underbrace{\theta_0}_{K\times 1} \quad \text{(linear)}$$

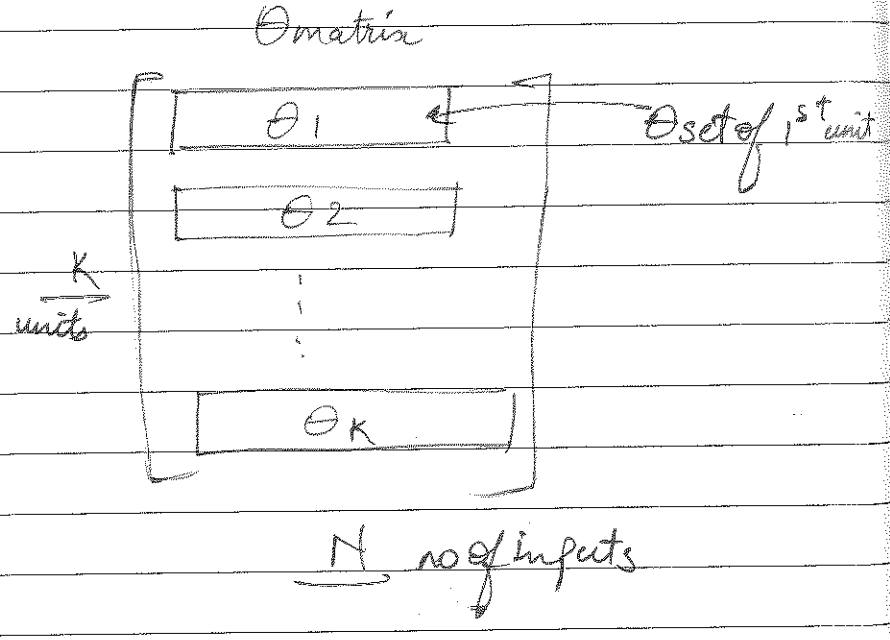
Neurons are in layers.



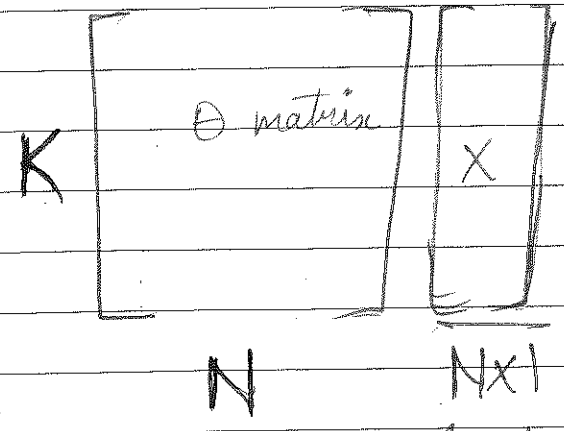


So,

$$\underset{K\times 1}{\hat{y}} = f(x;\theta) = f(\underset{K\times N}{\theta^T} \underset{N\times 1}{X} + \underset{K\times 1}{\theta_0})$$



So take product,



so far an entire layer we have,

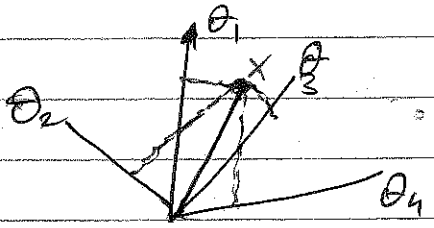
layer: $\theta_{K \times N}$ $\theta_{0\ K \times 1}$ $\hat{y}_{K \times 1}$

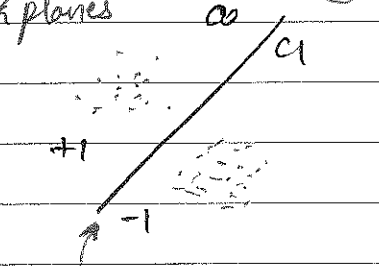
* Linear Classifiers interpretation

→ perform linear discrimination
each unit defines a linear decision boundary

→ perform template matching
each unit defines a template

→ perform projection into a new basis
each unit represents a basis vector





θ parameters of plane separating classes. (one for each unit)

- Note: in the end, its taking a weighted sum and passing it to activation function.

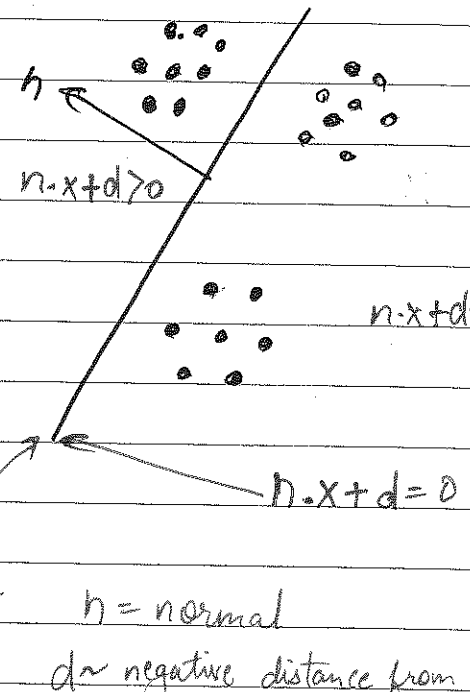
- each linear discriminant separates one class from all others.
- The value of a linear discriminant is +ve for examples belonging to class and negative otherwise. It measures distance from the decision boundary.

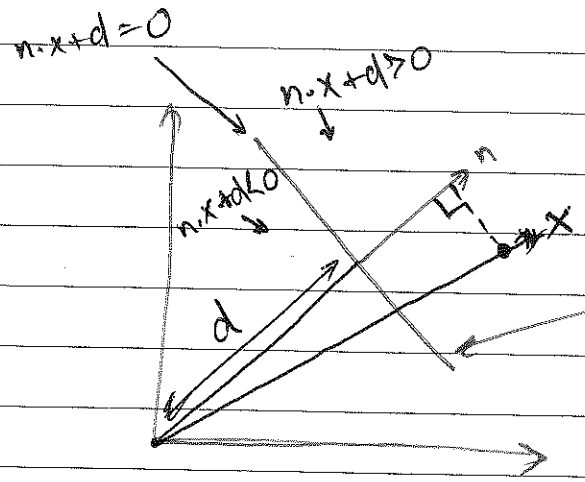
* just like hough; we have a discriminant function,

$$\hat{y} = \theta^T x + \theta_0$$

$$= \begin{bmatrix} n_1^T \\ n_2^T \\ \vdots \\ n_K^T \end{bmatrix} \begin{bmatrix} \\ x \\ \\ \end{bmatrix} + \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_K \end{bmatrix}$$

$$= \begin{bmatrix} n_1 \cdot x \\ n_2 \cdot x \\ \vdots \\ n_K \cdot x \end{bmatrix} + \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_K \end{bmatrix}$$





* Template matching

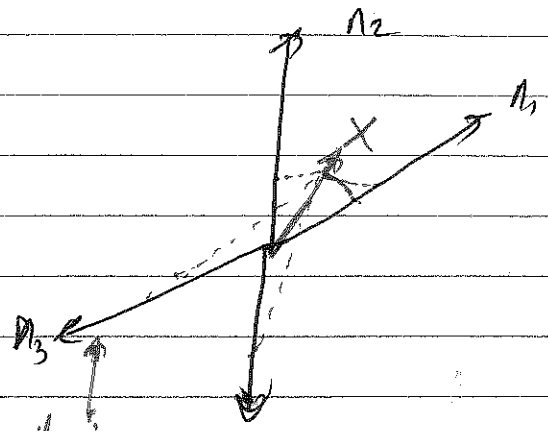
- rows of θ^T (rows of θ) are templates.
- with K rows of θ^T we have K templates (one templates per class)
- $\theta^T x$ measures how well x matches (dot product of x with rows of ~~with~~ θ^T).
- High similarity to a Template of a particular class

indicates high membership in this class.

$$\hat{y} = \theta^T x + \theta_0$$

$$= \begin{bmatrix} n_1^T \\ \vdots \\ n_K^T \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} d_1 \\ \vdots \\ d_K \end{bmatrix}$$

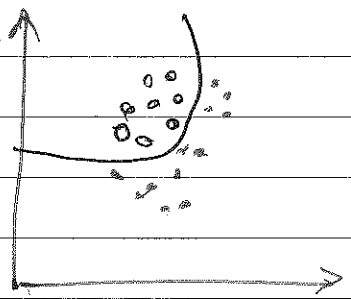
$$= \begin{bmatrix} n_1 \cdot x \\ \vdots \\ n_K \cdot x \end{bmatrix} + \begin{bmatrix} d_1 \\ \vdots \\ d_K \end{bmatrix}$$



* What if decision boundaries are non linear? What if more than one template is needed per class?

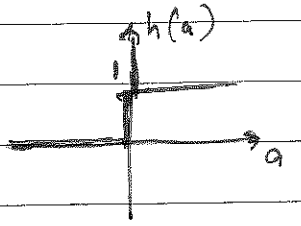
→ Combine multiple linear classifiers.

Its important to put activation function to these layers.

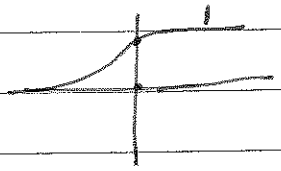


* Activation functions

- Step : $h(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$

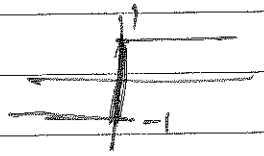


- sigmoid : $h(a) = \dfrac{1}{1+\exp(-a)}$

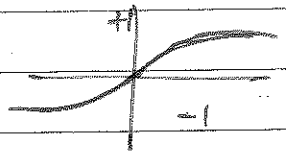


adv → it does not make sharp decision when its not so certain.

- Step* : $h(a) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{otherwise} \end{cases}$

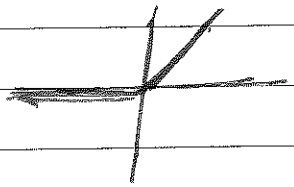


- tanh : $h(a) = \tanh(a) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

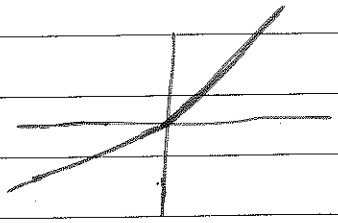


Good choice →

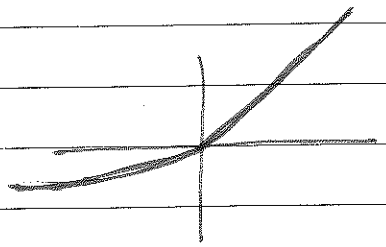
- ReLu : $h(a) = \max(0, a)$ (rectified linear unit)



- leaky ReLu : $h(a) = \max(\alpha a, a)$



- ELU : $h(a) = \begin{cases} a & a \geq 0 \\ \alpha(e^{a} - 1) & a < 0 \end{cases}$



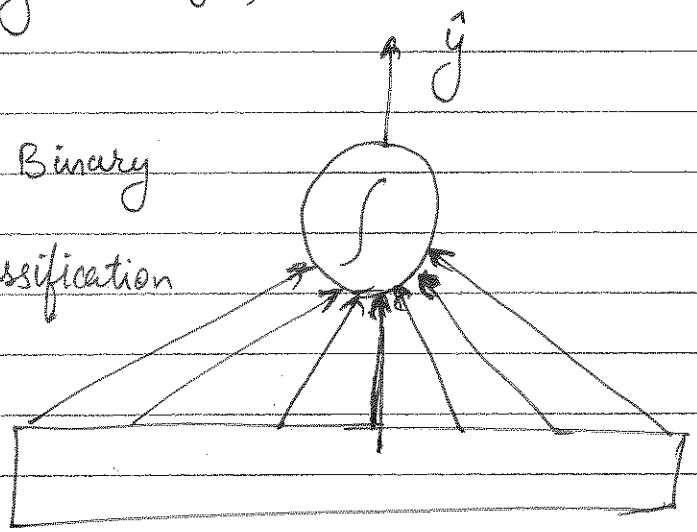
* Converting similarity to probability:

- Given similarity score (or decision boundary distance s_j) from a linear classifier (higher better), compute:

$$\hat{y}_j^{(i)} \equiv p(y=j \mid x^{(i)})$$

(i^{th} example; j^{th} unit; label; input)

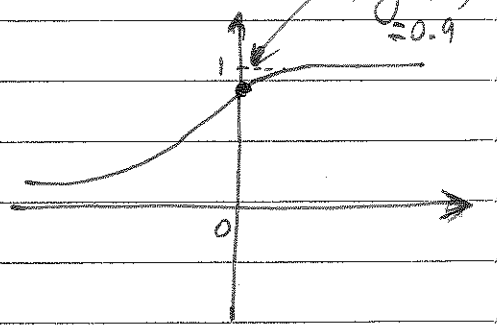
- For the last layer of the network use sigmoid($s_j^{(i)}$).



- In a 2 class classification case:

$$\hat{y}_j^{(i)} \equiv p(y=j \mid x^{(i)}) = \text{sigmoid}(s_j^{(i)})$$

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$



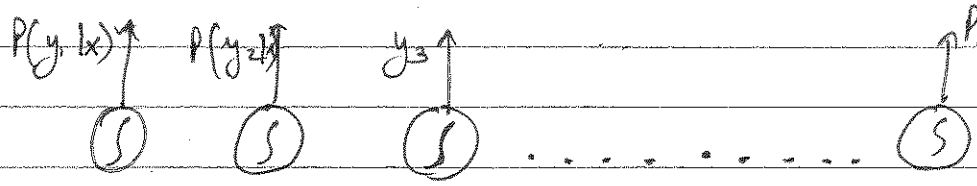
→ what happens in a K-class classification?

- Use softmax for K-class classification:

$$\hat{y}_j^{(i)} \equiv P(y=j \mid x^{(i)}) = \frac{\exp(s_j^{(i)})}{\sum_{l=1}^{K} \exp(s_l^{(i)})} \leftarrow \text{Softmax}$$

Lecture 15 - W821

(contd)



Since probability by taking the sum of their output would be >1 we need to normalize each outcome. So we take

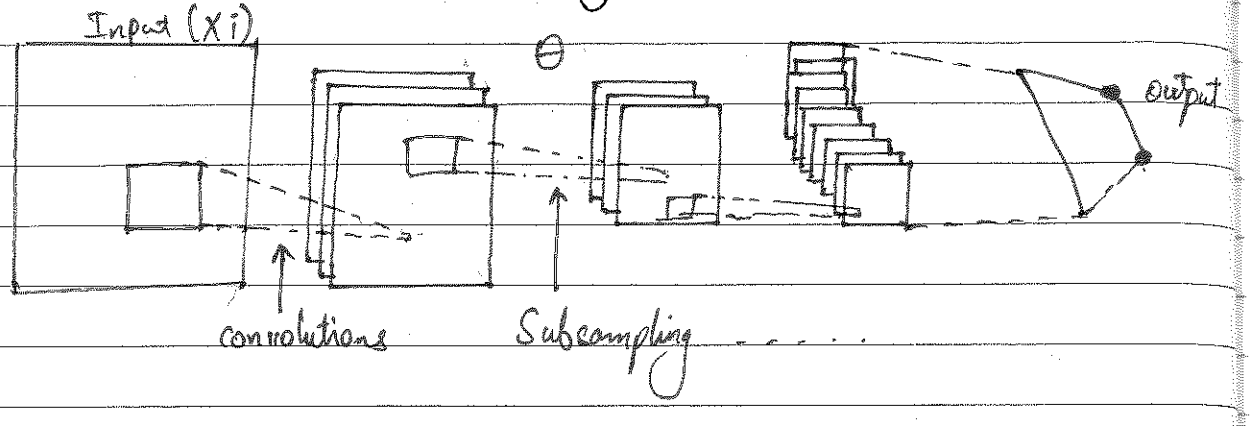
$$\frac{P(y=1|x)}{\sum P(y=j|x)}$$ as our outcome.

softmax

$$\hat{y}_j^{(i)} = P(y=j|x^{(i)}) = \frac{\exp(s_j^{(i)})}{\sum_{l=1}^{K} \exp(s_l^{(i)})}$$

(only used at output layer.)

* Network ~~Architecture~~ training :



training : find weights θ across layers.

input in image (x_i)

minimize difference between expected & obtained output :

$\Rightarrow$ input with labels $\{x(i), y(i)\}_{i=1}^{m}$

- Loss function $L(\theta)$ is difference b/w outcomes $\hat{y}(i) \longleftrightarrow y(i)$
- Optimization framework is used to minimize loss. by 'how to update properly'.

I)

Loss function :

- The weights θ are set to minimize loss :

$$\theta^* \equiv \underset{\theta}{\operatorname{argmin}}\ L(\theta)$$

$$\equiv \underset{\theta}{\operatorname{argmin}}\ \frac{1}{m} \sum_{i=1}^{m} L_i\left(x^{(i)}, y^{(i)}; \theta\right)$$

$$= \underset{\theta}{\operatorname{argmin}}\ \frac{1}{m} \sum_{i=1}^{m} L_i\left(f(x^{(i)}; \theta), y^{(i)}\right)$$

sample loss

$\hat{y}^{(i)}$ compare $y^{(i)}$

$\frac{1}{m}$: not exactly required

L1 and L2 Loss :

- In regression problems minimize ~~problems~~ distance between known and predicted values:

$$L1: L_i(\theta) = \sum_{j=1}^{k} \left| \hat{y}_j^{(i)} - y_j^{(i)} \right|$$

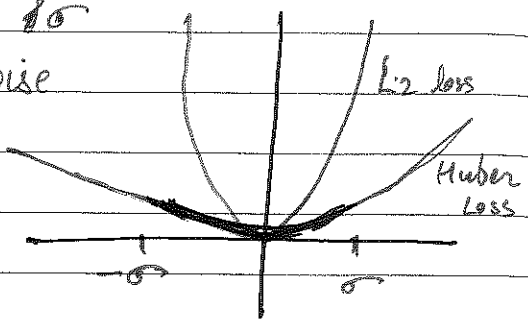
$$L2: L_i(\theta) = \sum_{j=1}^{k} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2$$

X	y	$\hat{y}$
-	3.2	-3.1
-	5.3	-5.6
-	7.2	9
-	⋮	⋮
⋮	⋮	⋮

$\hat{y}_j^{(i)}$ is predicted value : $y_j^{(i)} \in \mathbb{R}$

Huber Loss (like Geman - McClure)
- cap loss of outliers (robust regression) and also its easy to take derivative of this function

$$S_\sigma(d) = \begin{cases} \frac{1}{2} d^2 & \text{if } |d| \le \sigma \\ \sigma(d - \frac{1}{2}\sigma) & \text{otherwise} \end{cases}$$



- quadratic for small d
- linear for large d

$$\underline{L_i(\theta)} = \sum_{j=1}^{K} S_\sigma\left(\hat{y}_j^{(i)} - y_j^{(i)}\right)$$

Cross entropy Loss: (classification)
* convert similarity scores to probabilities:

$$\hat{y}_j^{(i)} = P(y=j \mid x^{(i)}) \qquad j \in [1, K]$$

Previous ones were for continuous outcomes, what if we want discrete outcomes (0, 1).

* Likelihood: $L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{K} \underbrace{\left(P(y=j \mid x^{(i)})\right)}_{\hat{y}_j^{(i)}}^{y_j^{(i)}}$ ($y_j^{(i)}$: 0, 1)

(multiply pred. prob. for correct class.)

↑ maximize

- Only the relevant $y_j^{(i)}$ outcomes are 1 and rest all become $x^0 = 1$, so the do not contribute.

* Problems with likelihood: ① We want to minimize loss & to be consistant with other loses. So we minimize -ve of likelihood. ② It is difficult to take derivative of product so we ~~use~~ need to change it to sum for which we use log.

* Log likelihood: $L(\theta) = -\log L(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{K} y_j^{(i)} \log\left[P(y=j|x^{(i)})\right]$

(-ve LL) minimize

$$L(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{K} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

cross entropy loss

where sample loss = $L_i(\theta) = -\sum_{j=1}^{K} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$ for single example

* possible cross-entropy loss values:

$L_i(\theta) \in [0, \infty]$ 0 when $P \to 1$, ∞ when $P \to 0$

- For random class assignment (worst case):

$$P(y=j|x^{(i)}) = \frac{1}{K} \to -\log\left(P(y=j|x^{(i)})\right) = K$$

$L(\theta) = \log(K)$ ← worst loss value

Regularization

- occam's razor : simpler explanations are better

↓

i.e. generalization because more complex ones can be overfitted & its not desirable.

- A simpler solution is when the weights θ are lower (eg when $\theta_{ij} = 0$ we remove one coefficient)

- smaller coefficients → more stable solution that will generalize better

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_i\left(f(x_i; \omega), y_i\right) + \lambda R(\theta)$$

- error term
- weight of regularization (hyper parameter)
- regularization term

where $R(\theta) = \sum_i \sum_j \theta_{ij}^2$

↑ L2 regularization

L1 regularization : $R(\theta) = \sum_i \sum_j |\theta_{ij}|$

II) Now that we have $L(\theta)$ loss fcnc value, we need to use optimization framework.

Optimization

- To minimize loss (objective) solve:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ L(\theta) \Rightarrow \nabla L(\theta) = 0 \Longrightarrow \theta^*$$

- If $\nabla L(\theta)$ is not linear it is hard to find an explicit solution. (The activation functions are non linear)

- So we use numerical iterative solution (eg. gradient descent).

Gradient Discent

- we guess the parameters, compute loss, then make second guess that's ~~better~~ based on first. loses gradient.

- start with initial guess θ_0

- repeat:

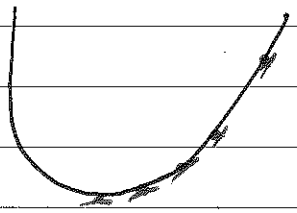
learning rate (hyper parameter)

gradient at previous solution

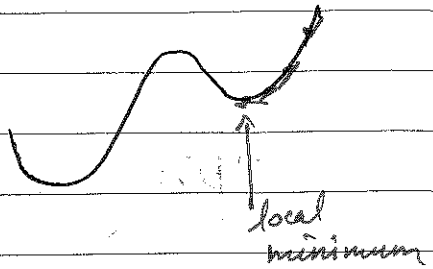
$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla L(\theta^{(i)})$$

- stop when

$$(L(\theta^{(i+1)}) - L(\theta^{(i)})) < \tau$$

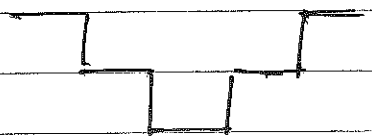


we might get local minimumand get stuck. so solution to these are available. (eg momentum.).



- The learning rate has to be selected correctly: if to big it overshoots. if its small barely moves but for good its fine.

- we cannot apply gradient discent to piecewise constant functions:



*Example:

- linear regression

$$\underset{1\times 1}{\hat{y}} = \underset{1\times n}{\theta^T} \underset{n\times 1}{x}$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}_{m\times n}$$

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(\overbrace{\theta^T x^{(i)}}^{\hat{y}^{(i)}} - y^{(i)}\right)^2$$

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(k)} \end{bmatrix}_{m\times 1}$$

$$= \frac{1}{2}\underset{1\times m}{\left(X\theta - Y\right)^T}\underbrace{\left(\underset{m\times n}{X}\underset{n\times 1}{\theta} - \underset{m\times 1}{Y}\right)}_{m\times 1}$$

$$\nabla L(\theta) = \frac{2}{2}\frac{1}{2} X^T(X\theta - Y)$$

- gradient descent will be:

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \underbrace{X^T}_{n \times m} \underbrace{(\overbrace{X\theta}^{\hat{y}} - y)}_{m \times 1}$$

$m \times 1$

$n \times 1$

$n \times 1$ scalar

- Summation form

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \sum_{j=1}^{m} \underbrace{(\overbrace{x^{(j)T}\theta}^{\hat{y}^{(i)}} - y^{(j)})}_{\text{scalar}} x^{(j)}$$

- update parameters by adding examples $x^{(j)}$ where the prediction is wrong.

Lecture - 16 (08L2)

Stochastic Gradient Descent (SGD)

epoch → traverse entire dataset

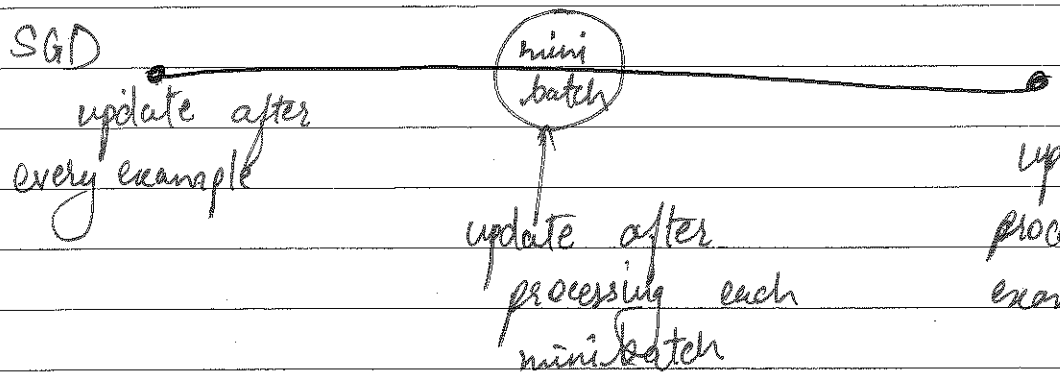
- Randomly order examples
- for $j = 1 \dots m$

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla L_j(\theta^{(i)})$$

(No sum; gradient based on example j)

- This converges faster.
  disadvantage is the gradient could be bad or wrong directional which is not desired.

- Thus in real world we never perform true SGD, but a ~~batched~~ hybrid is used.



Learning Rate Decay

- Learning rate need not be fixed
- make learning smaller as iterations progress

* Strategies:

1) Step decay:
   every K iterations $\eta \leftarrow \eta/2$

2) exponential decay — decay rate

$$\eta_t = \eta_0 \cdot e^{-k/t}$$ ← iteration index

3) fractional decay

$$\eta_t = \eta_0 / (1 + k t)$$

* Additional methods:

- Newton's method
  - compute learning rate instead of specifying it
  - second order derivatives (Hessian matrix)

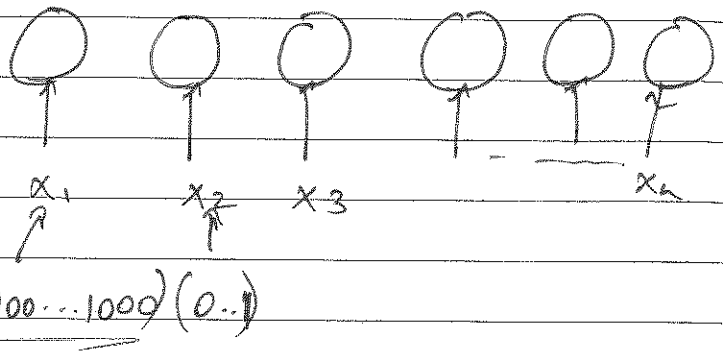
- AdaGrad:
  replaces Hessian computation (expensive) with approximation

- RMS Prop:
  - add decay to ~~AdaGrad~~ Adagrad

- ADAM:
  - add gradient momentum to RMS Prop

- Quasi-Newton methods (eg ~~DFGS~~)
  - approximate Hessian inverse (faster)

Data normalization



- features with high values have more influence and different features require a different learning rate.

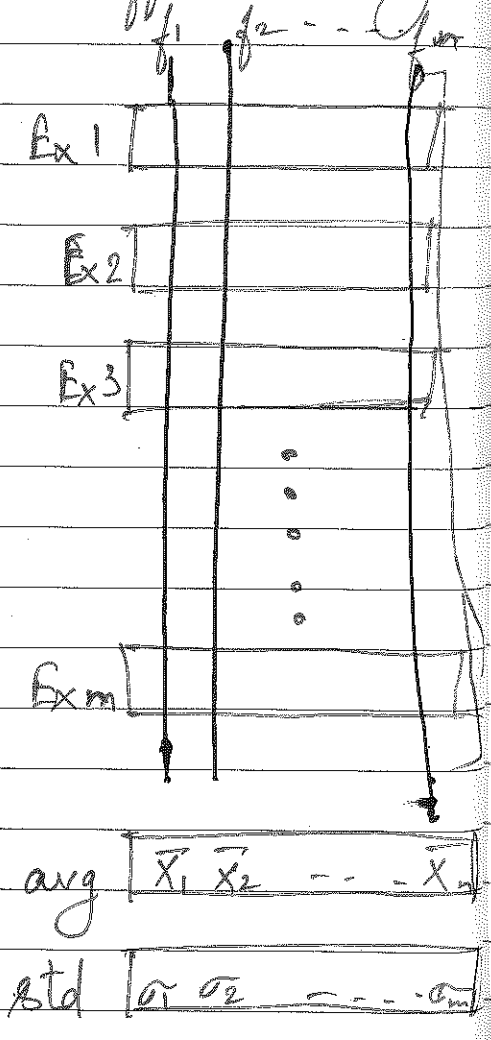
- Data normalization

$$\bar{x}_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

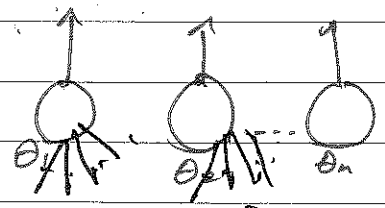
$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} \left(x_j^{(i)} - \bar{x}_j \right)^2$$

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \bar{x}_j}{\sigma_j}$$

normalized features

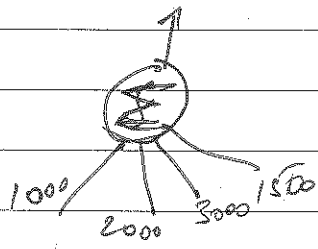


Weight Initialization



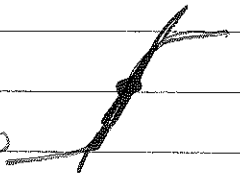
θ = constant → all outputs identical
→ no learning
→ random initialization

$\theta_i = [1, 2, 3, 4]$



θ - large → Saturated activation
→ Zero gradients
→ no learning
→ ~~initiated~~ initialize close to zero

→ Initialize at random close to zero



data metrics

* Glorot (Xavier) normal initialization
— Draw weights from normal distribution with:

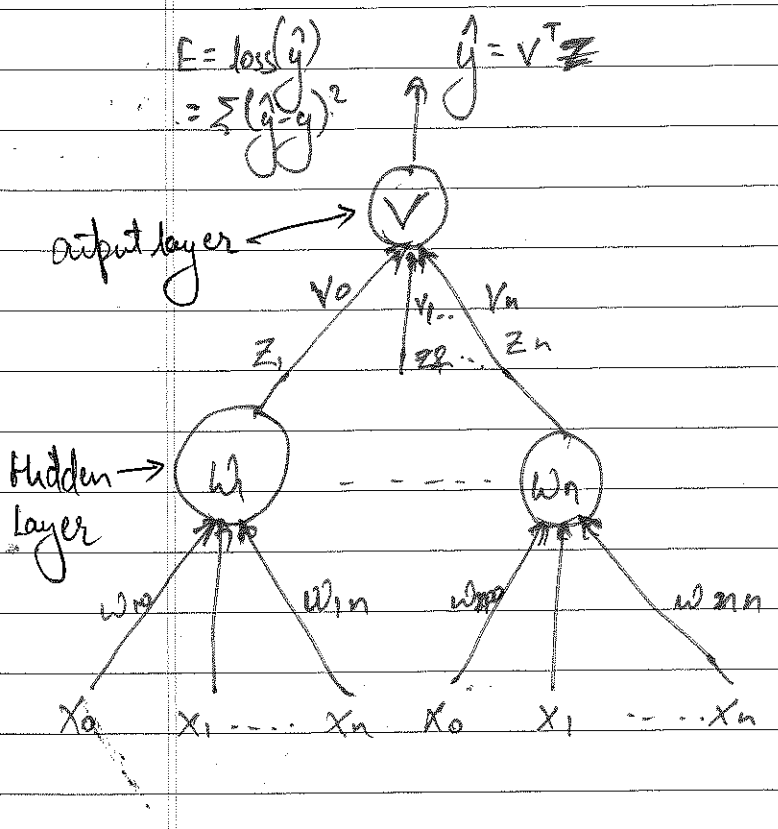
$$var[w_i] = \left(\frac{2}{fan\ in + fan\ out}\right)^{1/2} \qquad \mu = 0$$

fan in → Inputs
fan out → outputs

* Glorot uniform initialization
— Draw weights from uniform distribution

within $[-lim, lim]$ $\quad lim = \left(\frac{6}{fin + fout}\right)^{1/2}$

Backpropagation :



Chain rule :

loss → $\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v}$ → find v

Parameters (vector)

loss → $\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j}$

$\frac{\partial \hat{y}}{\partial z_j}$

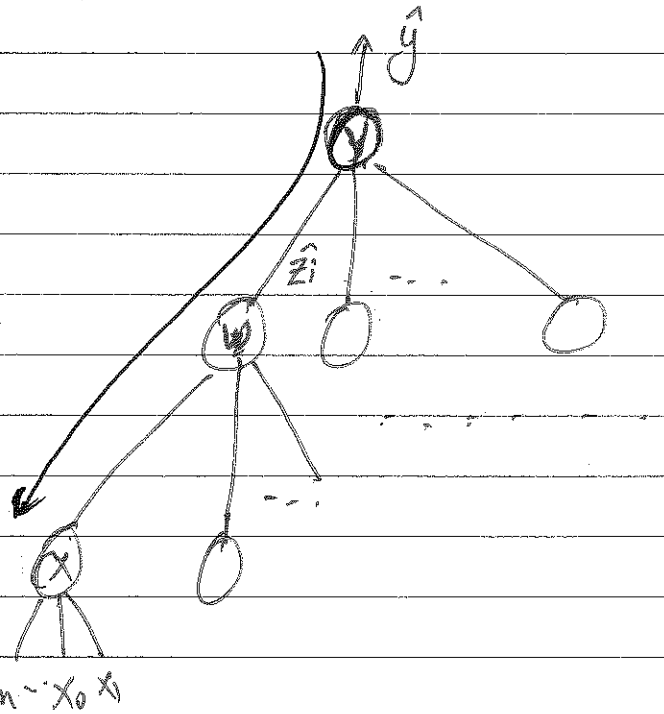
Parameter (vector) w_j

$\frac{\partial z_j}{\partial w_j}$

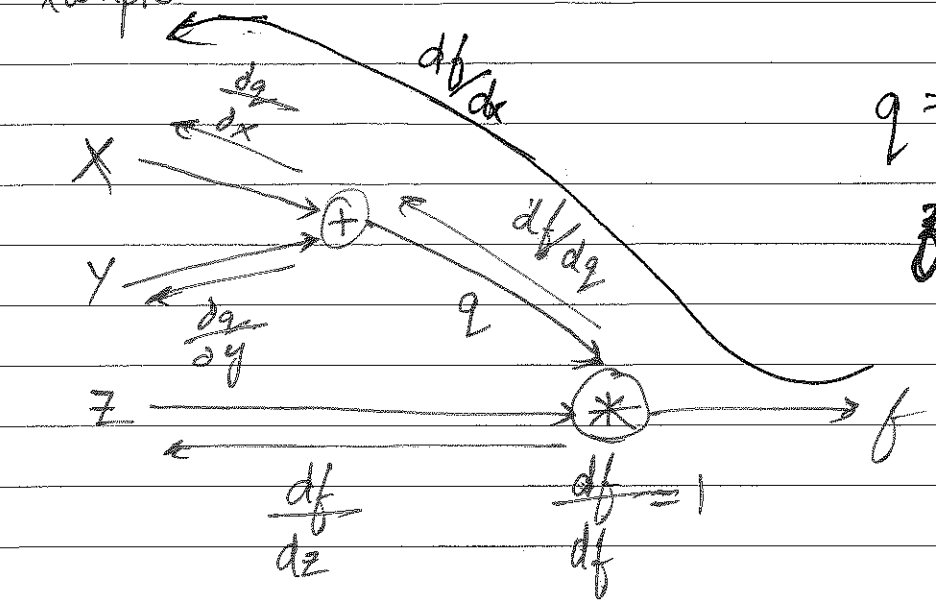
find w_j

- Start with guess for parameters : V, $\{w_j\}$ (at random close to zero)
- Use $x^{(i)}$ and current parameters to complete outputs : $z^{(i)}$, $\hat{y}^{(i)}$] forward pass / push example
- Use outputs $z^{(i)}$, $\hat{y}^{(i)}$ to update parameters : V, $\{w_j\}$
- continue while loss changes] backward pass (push back gradients)

- Represent network using a computational graph
- Because each node is simple, it has a simple explicit expression for its derivatives
- Forward pass: push input to compute all intermediate node values.
- Backward pass: starting with end nodes push gradients towards the beginning
- ~~Multiplying~~ Multiply backpropageted graddients (from back) by current gradients and propagate this.



Example:



Want : $\frac{df}{dx}$, $\frac{df}{dy}$, $\frac{df}{dz}$

(*) (+)

$\frac{df}{df} = 1$ $\quad \frac{df}{dq} = z$ $\quad \frac{dq}{dx} = 1$

$\frac{df}{dz} = q$ $\quad \frac{dq}{dy} = 1$

→ forward pass : push inputs to compute intermidia node values.

eg. $(x, y, z) = (1, 2, 3)$

$q = 1 + 2 = 3$

$f = 3 \times 3 = 9$

→ backward pass: ~~push inputs to compute intermediate node values.~~ starting at end push gradients towards beginning.

propagate ↓ computed by node ↓

$$\frac{\partial f}{\partial x} = \frac{df}{df} \cdot \frac{df}{dq} \cdot \frac{dq}{dx} = 3 \cdot 1 = 3$$

$$\frac{df}{dq} = z = 3$$

$$\frac{df}{df} = 1$$

$$\frac{df}{dy} = \frac{df}{dq} \cdot \frac{dq}{dy} = 3 \cdot 1 = 3$$

$$\frac{df}{dz} = q = 3$$

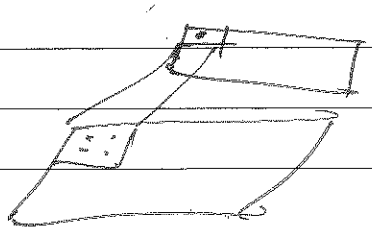
Regularization Methods (prev. covered)

Large number of parameters tend to overfit.
⇒ Need to prevent overfitting

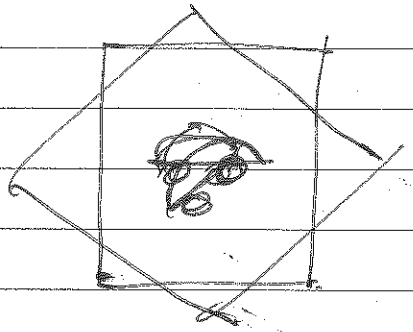
* Dropout:
  - at each training stage drop out units in fully connected layers with probability of (1-p), where p is hyper parameter of
  - Removed nodes are reinstated with original weights in the subsequent stages.

* Dropconnect:
  - Instead of dropping out unit, drop inputs with probability of 1-p (i.e randomness in inputs instead of output in normal dropout).

* stochastic pooling:
  - A 'dropout' version for convolution layers.
  - Activation within each pooling region is picked randomly according to a multinominal distribution.
  - Equivalent to normal pooling from a set of randomly deformed images (deformation increases exponentially with layers).



* Synthetic data: (data augmentation)
  - increase variability in training data:
    - synthesize new data
    - Perturb existing data (eg crop images in different ways.



* Early stopping:
  - stop training before learning completes.

* Reduce parameters:

  eg: reduce hidden layers.

* Weight decay:
  - multiply each coefficient by $\rho \in [0, 1]$

  As iterations progress weights that are not reinforced decay to 0. $(\rho)^{100} \to 0$

  - Equivalent to adding regularization term to loss function.

* Max norm constants:

- clamp to weights vector at each unit so that $||w||_2 < c$

* Batch normalization