Jan 28, 2021

Welcome To CMPE258 !

First Day of the Class

Harry LI, github/hualili/opencv/deep-Learning-2020S
20-2021S Email: hua.li@sjsu.edu

Office Hours M.W. 4:30-5:30 PM.
Zoom Based

(650) 400-1116 Text Only

On-Line material
- github/hualili
- CANVAS
  - Homework Assignment
  - Collect Submission of Homework

HANDS-ON:

Programming, Python

(1) OpenCV, T.F. (Tensor Flow) Keras API
google Colab

Action 1: Installation of OpenCV.
Version 4.2

Note: Use/Adopt Linux Ubuntu 18.04
VirtualBox

Note: Python3. Python Virtual Enviroment

3 major Areas
- (1) Handwritten Numerals Recognition. MNIST
  - C.V., R.O.I.
  - Deep NN
- (2) Time Series Prediction LSTM

↑ Subjects

(3) FaceNet, ResNet

(4) Deep Reinforcement Learning
DRL
Action — Policy — Reward

1/

Work to be done:

1. Programming / Code Development from Repo.
2. Write/Submit Pseudo Code (Brief Summary) Report 1 page
   Note: Post a Sample on github Latex
3. Homework Submission
4. Submission (Including Semester Long team Project).

Action 2: Form 4-person Team By Feb 14 week; work has to Individual / Encourage team Discussion.

Grading Policy:
- Mid: 30%
- Homework: 30%
- Final: 40%

* Introduction

Neural Nets
Biological System
Human Brain
Neurons (Cells)

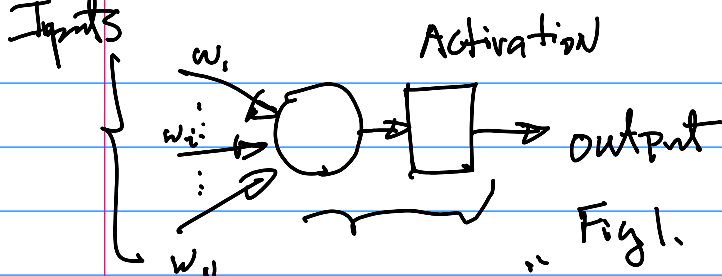


Fig 1.

Prof. Cavard Mead — "Silicon Brain"

Intel Processor ~1992–94, "Father of VLSI"

~1994–95

Autonomous System

OpenCV 1997

2005–2006

Standford Group + Google

Self Driving Market Huge

2013 AlexNet ("Deep" convolutional Neural Network)

NVDA GPU Architecture

Recently: FaceNet, ResNet

Time Series Analysis (LSTM)

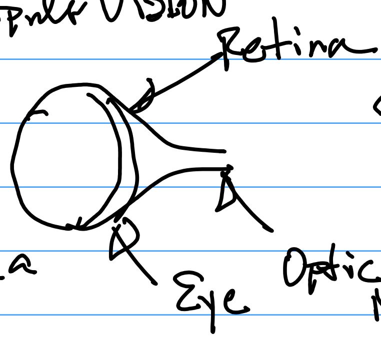
Deep Reinforcement

Computer Vision

multiple Layers of Photo Receptors

~10 layers.

110 ~ 120 million P.R.



(Brain cells) ~1 million

Fig. 2a

Image Pyramid

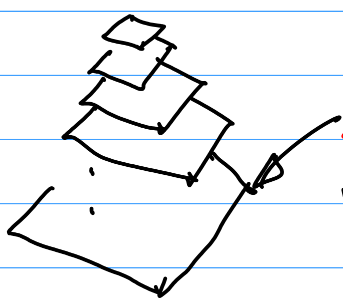


Fig. 2b

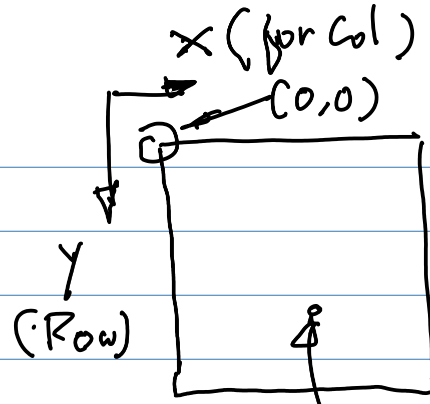


Fig. 3

I(x,y)

M×N

Resolution

Note:

1° "Scanning" ~ From L to R, top to B

"L2R, T2B"

2° Resolution 1024×768

No. of colum — No. of Row

M×N → 1024×768

M — Colum — X

N — Row — y

3° Color Image — Vector

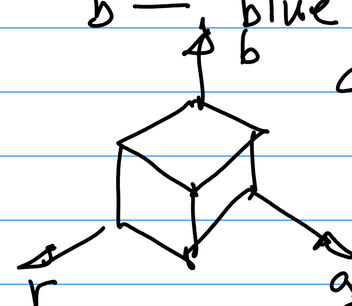
Color Space (r, g, b)

Primitive Color

r — red;

g — green;

b — blue



color Cube

Fig. 4

(0,0,0) Black (1,1,1) White

grey scale as Traveling from (0,0,0) towards (1,1,1)

Highest Red $(r,g,b) = (1,g,b)$

" Green $(r,g,b) = (r,1,b)$

" Blue $(r,g,b) = (r,g,1)$

$I(x,y)$ → Intensity in terms of (r,g,b)

① Location On the Image Plane

$I(100, 201)$

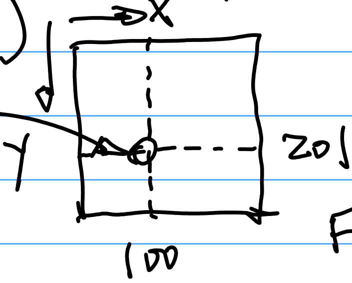


Fig5

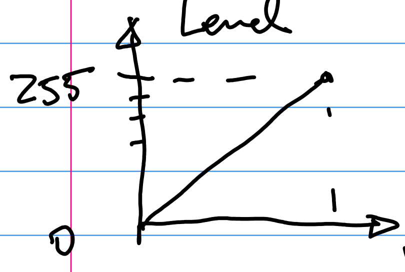
r: 8 bit [0,255] $(2^8 = 256)$

g, b: 8 bit, " "

Quantization Level

Pixel Depth (BPP: Bit Per Pixel)

For 8 bit



Grayscale Image

$$I(x,y)_g = \frac{1}{3}\left[r(x,y) + g(x,y) + b(x,y)\right] \quad \ldots (1)$$

Convert Color Image to Gray Scale Image

Action 3. Enable OpenCV {colab, Locally (Install)}

.... Display A "Jpg" Color Image → From Your Smart

Feb 4th, CMPE258

Today's Topics: 1° Convolution, Two Dimensional Convolution; 2° Intro to Neural Network

Note: 1° Installation of Python, OpenCV, and T.F.

Python {2.7, 3.5 or higher}

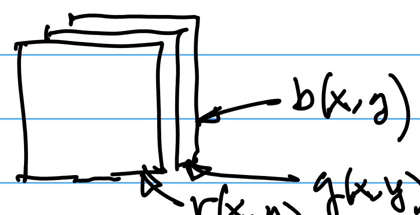
Anaconda — Python Distribution → Python Virtual Environment

OpenCV 4.2 Version 3.0 OR higher

T.F. Keras (API)

Given a digital image

$I(x,y) \rightarrow r(x,y), g(x,y), b(x,y)$



Example: Suppose an $I(x,y)$ is given below,

x →, Y ↓

0	2	3	10	0
0	2	3	0	0
0	2	3	0	0
0	2	3	0	0
0	2	3	0	0

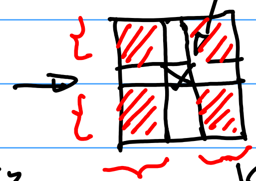
M×N

Perform Convolution on $I(x,y)$

Convolution Kernel (mask)

$$\begin{array}{c|c} & x \rightarrow \\ \hline y \downarrow & \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \end{array}$$

3x3 →



Ref: [_2025_...2DConvolution

From 1D Case, given a function $f(x)$ and a Kernel $g(x)$

$$\int_\Omega f(u)\,g(x-u)\,du \quad \dots (1)$$

$$\sum_{i\in\Omega} f(i)\,g(x-i) \quad \dots (2)$$

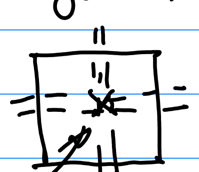
Now, 2D Case

$$\iint_\phi f(u,v)\,g(x-u,y-v)\,du\,dv \quad \dots (3)$$

$$\sum_j \sum_i \underbrace{f}_{\text{Image}}(\underbrace{i,j}_{\text{Index of pixel location}})\,\underbrace{g}_{\text{Kernel}}(x-i,y-j) \quad \dots (4)$$

Kernel $g(x,y) \rightarrow g(-x,-y)$

① Flip



→ a) flips the Kernel

b) $g(x,y) = g(-x,-y)$ if Symmetric

② $f(i,j)\,g(x-i,y-j)$ product

③ $g(x-i,y-j)$ shift

$g(1-i,1-j)$

$g(2-i,1-j)$

$g(2-i,2-j)$

and so on ...

Summary: 2D convolution consists of shift-product-summation

Compute 2D convolution:

Step 1. Place the kernel @ the initial condition, e.g., the top left Hand corner of the image, in such a way its boundary rows and columns are aligned with the image boundary row and colum.

$f(0,0)*1 + f(1,0)*0 +$
$f(2,0)*(-1) +$

$f(0,1)*1 + f(1,1)*0 +$
$f(2,1)*(-1) +$

$f(0,2)*1 + f(1,2)*0 + f(2,2)*(-1)$

$= 0*1 + 2*0 + 3*(-1) +$
$0*1 + 2*0 + 3*(-1) +$
$0*1 + 2*0 + 3*(-1)$

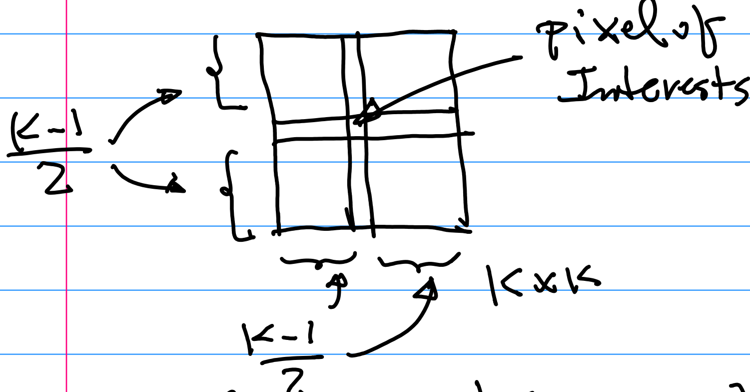
$= -3 - 3 - 3 = -9$

Note: This convolution resulted in a new processed image plane $I_{New}(x,y)$ whose Rows is less then the orginal image,

No. of Rows = Original No. of Rows − 2

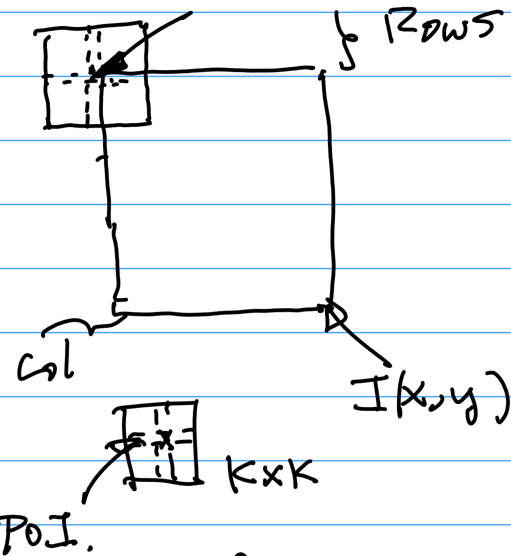
for 3x3 kernel, for KxK kernel (K is odd Number), in this case No. of Rows Reduced by $2*\left(\frac{K-1}{2}\right)$

$= K-1.$



Homework (Exerise — No Submission)
Based on the given $I(x,y)$ Image from PPT @ github, Perform hand Calculation of 2D convolution

Note follow the example in Class.



Consider Neural Networks.
Supervised Learning
Reference: 20-2021S-2

Example:

Input $(x_1, x_2, \dots, x_n) = \bar{x}$

Weights $(w_1, w_2, \dots, w_n) = \bar{w}$

$\bar{x} \cdot \bar{w} = (x_1, x_2, \dots, x_n) \cdot (w_1, w_2, \dots, w_n)$

$= x_1 w_1 + x_2 w_2 + \dots + x_i w_i + \dots + x_n w_2$

$= \sum_{i=1}^{n} x_i w_i \quad \dots (1)$

Define Transfer function $f(\cdot)$ as follows

$$f(\bar{x}) = \begin{cases} 1 & \sum_{i=1}^{n} x_i w_i + b > 0 \\ 0 & o/w \end{cases} \quad \dots (2)$$

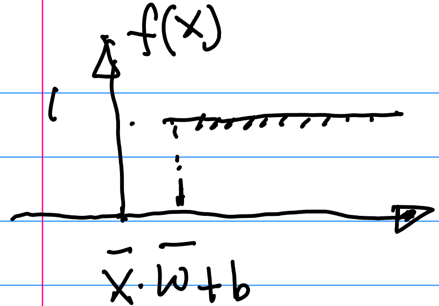


Fig 1.

Consider a Simple Feedforward NN

Example:

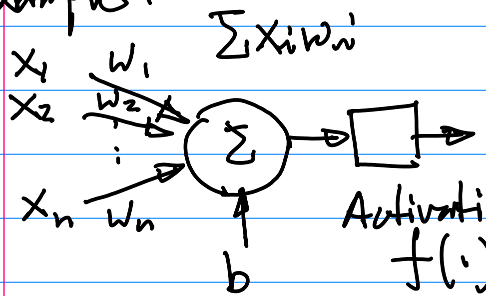


Fig. 2

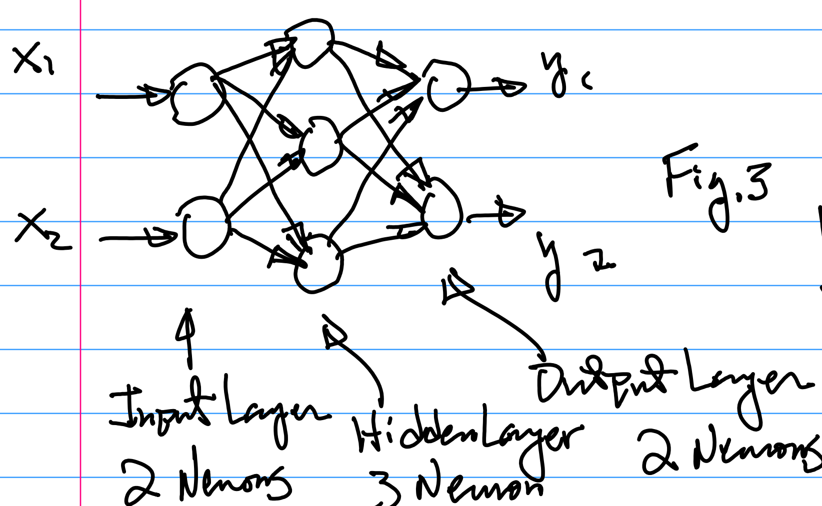


Fig. 3

FeedForward

Have Training Dataset Φ or Ω

Two Classes C_1, C_2 Representing 2 Patterns

Feature Vector $X = (x_1, x_2)$, up to N_1 of them for C_1, up to N_2 for C_2

N_1, N_2 don't have to be equal; Training (Learning)

$$w_i^+ = w_i^- + \Delta w_i = w_i^- + \eta(d - y)x_i \quad \ldots(3)$$

updated, Current, Rate, desired Output, Actual Output

Feb 11,

Today's Topics: 1° Introduction to NN; 2° Coding (Python) Examples.

Action 1: Form 4-Person Team. Submit your team member information By Friday 5:00 pm.

Subject: CMPE258 First Last Name of the group coordinator

First, Last Name, Last 4 Digits of your SID; E-mail Contact information;

Homework 1: Due A week from Today; Submission to CANVAS.

Consider Dense Feedforward NN From the PPT.

1. A Single Neuron {Block Diagram, Notation}

$\bar{x} = (x_1, \cdots, x_2)$

$y = f(\bar{x}) = f(x_1, x_2, \cdots, x_n)$

Activation function

$$z, S(x) = \frac{1}{1+e^{-x}}$$

Sigmoid

3. Python Implementation
Create A template for your program header,

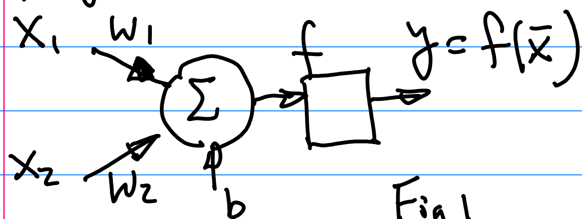


Fig 1.

$X_1 = 2, X_2 = 3, W_1 = 0, W_2 = 1.$
$b = 4, \; f(x) = \text{Sigmoid}(x)$

4. Feedforward NN

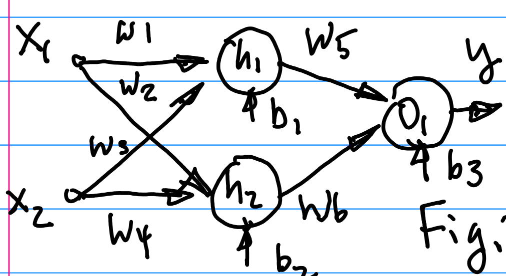
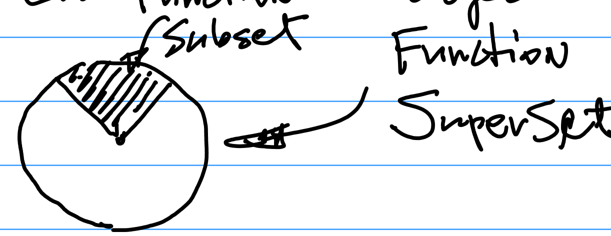


Fig.2

5. Data Set → Pre-processing to allow Activation function to Better handle the input values

Supervised Learning
Data → Its Categories Known
Labeling Images/Videos → Annotation.
ImageNet ~1.4 million Test Images

6. Loss Function → Objective Function

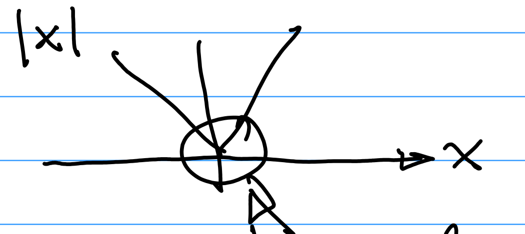


$y_{true}, \; y_{pred}$ — Notation & meaning

y_{pred} gives the information for which Category the data belongs to

7 Use squared difference to handle potential error (Loss) Cancellations due to opposite Signs.

Note: Be Careful Not Absolute value!



No derivatives at this point.

8. Define Loss function
↓
Behavior of Loss Function → minize the Loss/error function in the negative direction of the gradient

Gradient gives the fastest increase of the error

9, Denote the Loss as

$$L(W,b) = \sum (y_{True} - y_{Pred})^2 \cdots (2)$$

$b = (b_1, b_2, \dots, b_M)$

$W = (w_1, w_2, \dots, w_N)$

$M \neq N$

θ CMPE258 Feb18

Reference: 1° Training Algorithm (PART I) 2° OpenCV — Preprocessing

Objectives: { Deep Convolutional NN, Deep Reinforcement Learning }

① Handwritten Digits Recognition MNIST ② Facial Recognition ③ Deep Reinforcement Learning ④ LSTM Time Series Data

Note: 1° Minimization, Optimization, Maximization.

$f_{min} \rightarrow 1/f_{max}$?

$g_{max} \rightarrow 1/g_{min}$

2° Taylor Expansion

Basic Building Blocks to Build A Given function f(x) provided it has derivatives up to order K.

$$\lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x} = f'(x) = \frac{d}{dx} f(x) \cdots (1)$$

$$\Delta x \rightarrow \Delta f \rightarrow \Delta f / \Delta x \rightarrow \frac{\Delta f}{\Delta x}\Big|_{\Delta x \to 0}$$

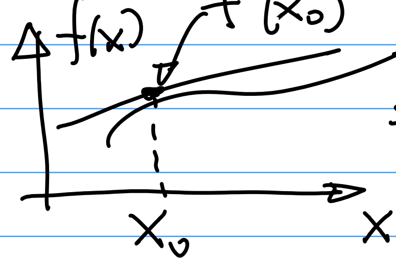


Fig.1

$f'(x_0) > 0 \rightarrow f(x_0 + \Delta x) - f(x_0) > 0 \cdots (2)$

Increase

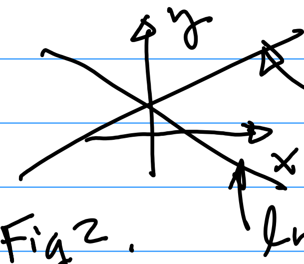
$f'(x_0) < 0 \rightarrow f(x_0 + \Delta x) - f(x_0) < 0 \cdots (2*)$

$f(x) = \text{BasicBuildingBlock}_0 + BBB_1 + BBB_2 + BBB_3 + \cdots \cdots (3)$

Higher order terms

BBB_0 1st One Constant

BBB_1 2nd one Linear

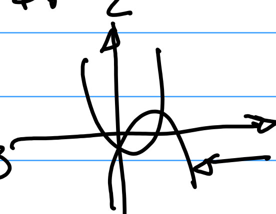


$y = ax + b$

Fig 2.

BBB_2 3rd 2nd Order Quadratic

Fig 3



$y = ax^2 + bx + c$

BBB_3 4th 3rd order Cubic

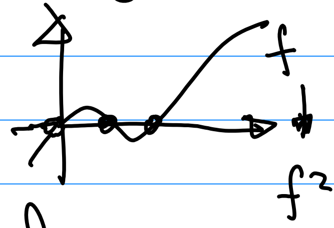
$y = ax^3 + bx^2 + cx + d$

CMPE258

9/

$$f(x)=f(x_0)+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots \quad (3^*)$$

Fig 4



Root Finding
Minization

3° For N-Dimensional Case

$$\underbrace{w_1, w_2, \cdots, w_{N_1}; b_1, b_2, \cdots b_m}_{N=N_1+M}$$

Rewrite $b_1, b_2, \cdots, b_m$ as w_i

$i=N_1+1, N_1+2, \cdots, N_1+M$

$L(w,b)$

$\downarrow$

$L(\underline{\underline{w}})$

Partial Derivatives

One-Dimensional Case $f'(x)=\frac{d}{dx}f(x)$

N-Dimensional Case

$$\frac{\partial}{\partial x_1}f(x_1,x_2,\cdots,x_N)$$

$$\frac{\partial}{\partial x_2}f(x_1,x_2,\cdots,x_N)$$

$$\cdots$$

$$\frac{\partial}{\partial x_N}f(x_1,x_2,\cdots,x_N) \quad \ldots(4)$$

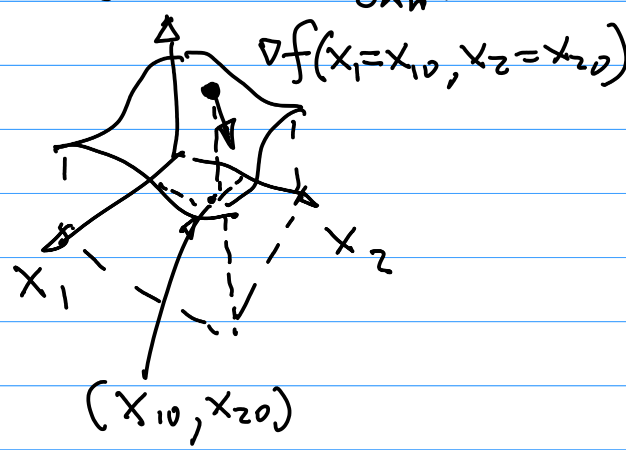
Find $x_i^{n+1}=x_i^n+\Delta x_i^n$

$$f(x_i^{n+1})-f(x_i^n)<0 \quad \ldots(5)$$

$\Delta x_i^n = ?$

4° Gradient

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \ldots(6)$$



5° Update Rule

$(x_1^{n+1}, x_2^{n+1}, \cdots x_n^{n+1})$ New updated Values

Based on Gradient with "—"

$$-\eta\nabla f = -\eta \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \ldots(7)$$

$$f(x_1, x_2, \cdots, x_n) = f(a_1, a_2, \cdots, a_n) + \frac{\partial f}{\partial x_1}(x_1 - a_1) + \frac{\partial f}{\partial x_2}(x_2 - a_2) + \cdots + \frac{\partial f}{\partial x_n}(x_n - a_n)$$

$+$ higher Order Term $\quad \cdots (8)$

Remove higher Order terms,

$$f(x_1, x_2, \cdots, x_n) \simeq f(a_1, a_2, \cdots, a_n) + \frac{\partial f}{\partial x_1}(x_1 - a_1) + \frac{\partial f}{\partial x_2}(x_2 - a_2) + \cdots + \frac{\partial f}{\partial x_n}(x_n - a_n) \quad \cdots (8*)$$

$$\underbrace{f(x_1, x_2, \cdots, x_n) - f(a_1}_{\text{"} < 0 \text{"}}, a_2, \cdots, a_n) = \cdots$$

$$f(x_1, x_2, \cdots, x_n) < f(a_1, a_2, \cdots, a_n) \quad \cdots (9)$$

We want RH $=$

$$\frac{\partial f}{\partial x_1}(x_1 - a_1) + \frac{\partial f}{\partial x_2}(x_2 - a_2) + \cdots + \frac{\partial f}{\partial x_n}(x_n - a_n)$$

$$(\underbrace{x_1 - a_1}_{\Delta x_1}, \cdots, \underbrace{x_n - a_n}_{\Delta x_n}) \cdot \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \cdots (10)$$

Gradient

from Eqn (11) (Handout, Note1, pp.1)

Let $\Delta x_1 = -\frac{\partial f}{\partial x_1}, \cdots,$

$$\Delta x_n = -\frac{\partial f}{\partial x_n} \quad \cdots (11)$$

Substitute Eqn (11) Back to Eqn (10)

$$(\Delta x_1, \Delta x_2, \cdots, \Delta x_n) \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

$$= \left(-\frac{\partial f}{\partial x_1}, -\frac{\partial f}{\partial x_2}, \cdots, -\frac{\partial f}{\partial x_n}\right) \begin{vmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{vmatrix}$$

$$= -\underbrace{\left[\left(\frac{\partial f}{\partial x_1}\right)^2 + \cdots + \left(\frac{\partial f}{\partial x_n}\right)^2\right]}_{\text{"} \leq 0 \text{"}} \quad \cdots (12)$$