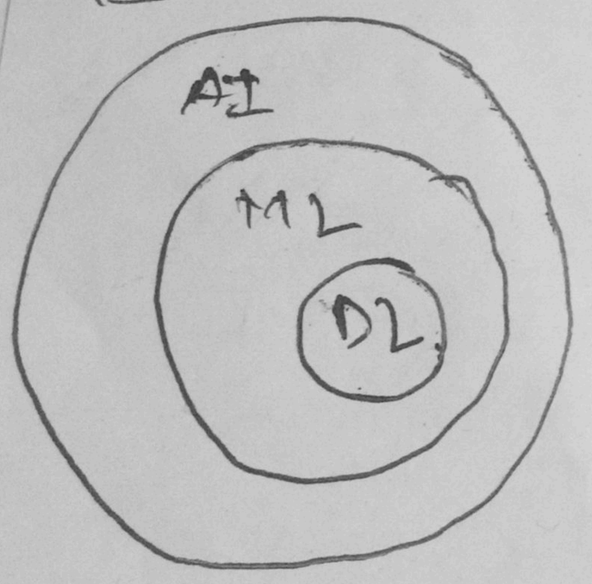
#Deep learning

Artificial Intelligence
↳ Theory and development of computer systems able to perform tasks normally require human intelligence
↳ Anything that includes in ML, DL

Machine Learning
↳ Gives computer a "the ability to learn without being explicitly programmed"
↳ SVM, KNN, RF, LinR, LogR, NB etc. Rule based

Deep learning
↳ ML algorithms with brain-like logical structure of algorithms called artificial neural networks
↳ NN, CNN, RNN, LSTM, RL, Transformer etc. learning based



Deep learning Timeline (History of DL)

1940 → Dark Era (Until 1940)
1943 → Neural Nets (McCulloch & Patt) (study about Biological neuron)
1950 → Computing Machinery and Intelligence (Alan Turing)
1958 → Perceptron ~~(Reception FF)~~ (Rosenblatt)

(Adaptive linear elements)

1960 → ADALINE (~~Widrow~~ Widrow & Holf)

1969 → XOR problem (Minsky & Papent) [improved]

1974 → Backpropagation (Werbos and Mone)

1980 → ~~Neocognitron~~ Neocognitron (Fukushima)

↳ Self Organizing Map (Kohonen)

1982 → Hopfield Networks (John Hopfield)

1985 → Boltzmann Machine (Hinton ~~&~~ & Sejnowski)

1986 → Restricted Boltzmann Machine (Smolensky)

↳ Multilayer Perceptron (Rumelhart, Hinton & Williams)

↳ RNNs (Jondan) (Recurrent Neural Network)

1990 → LeNet (Leeun) → meta (AI scientist)

1997 → LSTMs (Hochreiten & ~~Schmid~~ huber) [long short term memory]

↳ Bidirectional RNN (Schuster & Paliwal)

2006 → Deep Boltzmann Machines (Salak hutdinov & Hinton)

↳ Deep Belief Networks - pretraining (Hinton)

2012 → Dropout (Hinton)

2014 → GANs (Good fellow)

2017 → Capsule Networks (Sabour, Frosst, Hinton)

AI Winter

ILSVRC - 2012

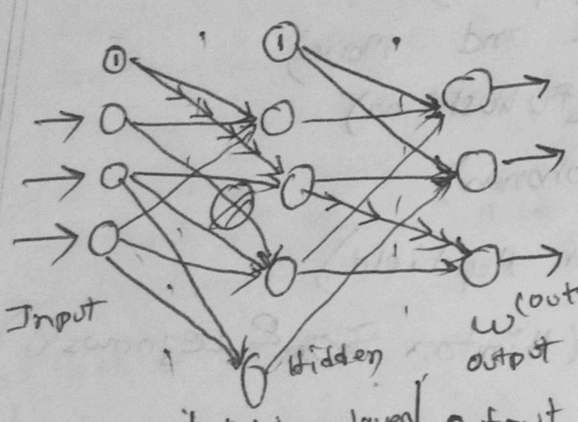
AlexNet - 2012 (ImageNet)

Hinton

GPU (2011-12)

Multi layer perceptron, 1986 (David Rumelhart)
↳ Multiple Neuron

[Neural Networks
Engine = Backpropagation]



(1st Input to 2nd hidden layer to the 3rd layer of output)

Input layer with 3 input units plus bias units ($m = 3+1$)	Hidden layer with 4 hidden units plus bias unit ($d = 4+1$)	Output layer with 3 output units ($t = 3$)

Number of layers, $L = 3$

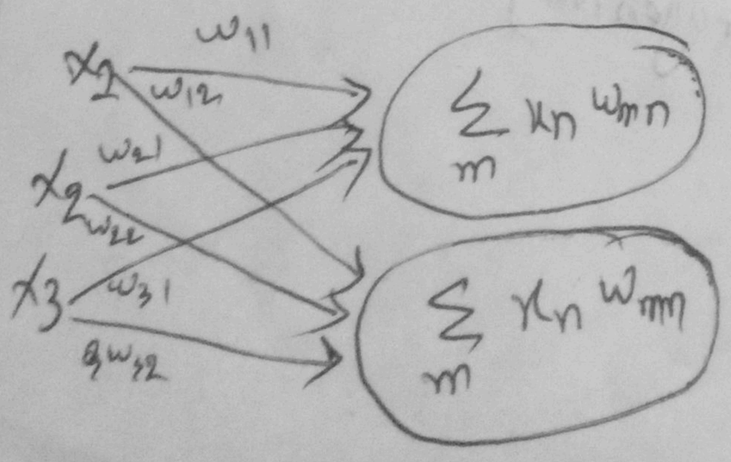
Non-linear units (how to train?)
↳ Hinton picked it up → baeks propagation (1974) chapter → Engine of Neural Networks
↳ to train MLP

He published Nature paper

Simple stacks of perceptron?

MLP sounds good?

Components of MLP, Linear function



function output

$$z_n = f(x_m, w_{mn}) = b + \sum_m x_m w_{mn}$$

- f: function name
- (x_m, w_{mn}): function inputs
- b: bias term
- m: index for each neuron and matrix row
- n: element of feature vector (x)
- w_{mn}: element (m, n) for feature matrix (W)

$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $W = \begin{bmatrix} w_{1} \\ w_{2} \\ w_{3} \end{bmatrix}$

$z = W^T \times x = [$

Sigmoid function
↳ An Activation function (a)

function output → $a_m = \sigma($

(Activation function)

Non-linearit

Output functi

$\hat{y} = f(a_m)$

$\hat{y} = f(a_m$

$\hat{y} = \sigma$

Cost fun

sum of squ

errors

the entir

dataset

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

$w_{11}\ w_{12}\ w_{13}$
$w_{21}\ w_{22}\ w_{23}$
2×3 $\qquad$ 3×1

$$z = W^T \times x = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 w_{11} + x_2 w_{12} + x_3 w_{13} \\ x_1 w_{21} + x_2 w_{22} + x_3 w_{23} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Sigmoid function
↳ An Activation function (a)

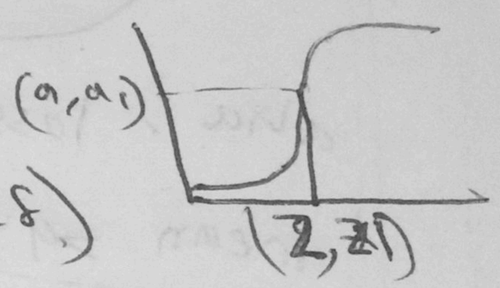
linear function output becomes sigmoid input

function output → $a_m = \sigma(z_m) = \dfrac{1}{1+e^{-z_m}}$

(Activation function)

sigma (sigmoid function)

Eulen's number (≈ 2.71828)



Non-linearity → (to separate Non-linear MLP)

Output function

$\hat{y} = f(a_m) \begin{cases} +1, & \text{if } a > 0.5 \\ -1, & \text{otherwise} \end{cases}$ — Threshold function, Binary classification

$\hat{y} = f(a_m) = a_m$ — Identity function, Regression problem

$\hat{y} = \sigma(a)_i = \dfrac{e^{\beta a_i}}{\sum_{j=1}^{K} e^{\beta z_j}}$ — Softmax function, Multiclass classification

Cost function

Sum of squared errors for the entire dataset

$$E = \frac{1}{2} \sum_{k} (a_k - y_k)^2$$

added to simplify gradient computation

expected value for training example k

index for input-output pair

predicted value for training example k

Cost function

For training example y
predicted value, expected value

$$E = \left(\frac{1}{2}\right) \sum_{y} (a_y - y_y)^2$$

added to simplify gradient computation

y → index for input-output pair

aka: loss function, cost function, Objective function, general function to optimize

mean squared error, sum of squared error, binary cross-entropy

↑ parameters

Forward propagation (Forward pass)

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

matrix multiplication
input to hidden layer

linear function

$$z_m = b + \sum_m x_n w_{mn}$$

sigmoid function

$$\sigma_m = \sigma(z_m) = \frac{1}{1 + e^{-z_m}}$$

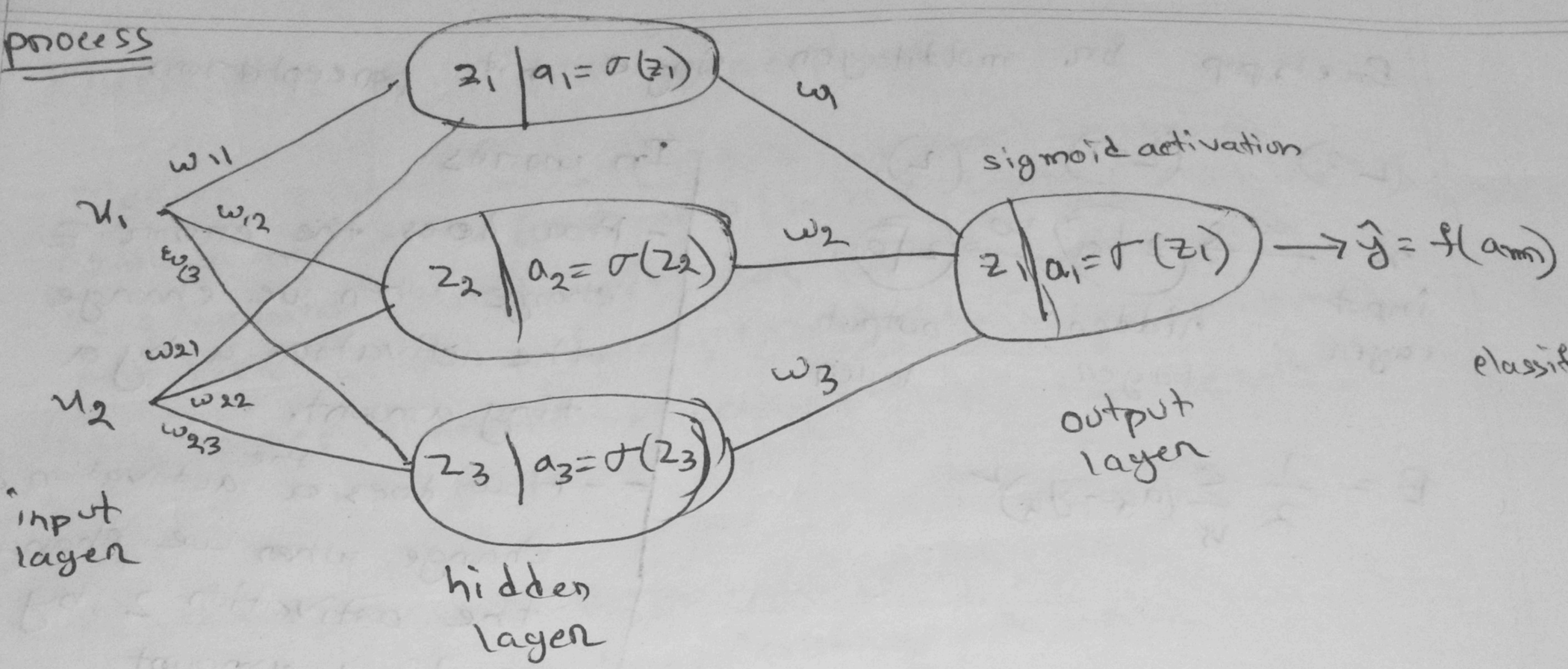
matrix multiplication
hidden to output layer

$$[w_1 \; w_2 \; w_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

decision threshold

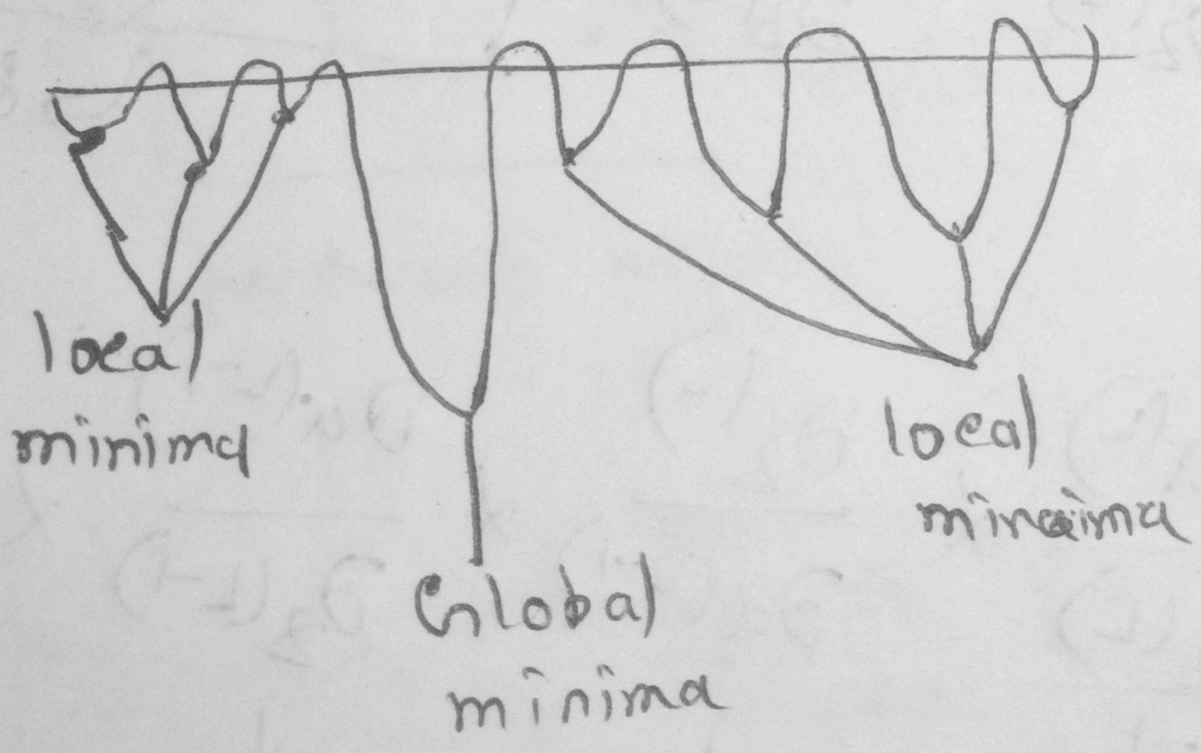
$$f(a_m) = \begin{cases} +1 & \text{if } a > 0.5 \\ -1 & \text{otherwise} \end{cases}$$

process

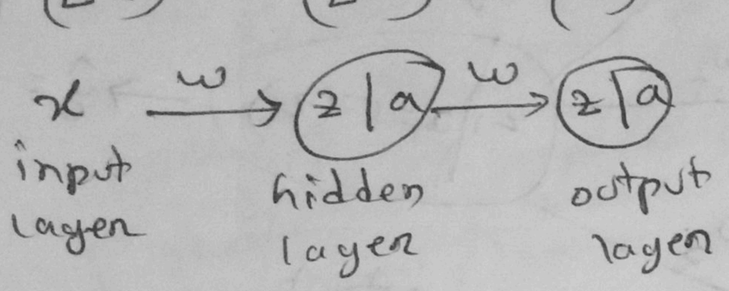


Backpropagation algorithm

- minimize the error gradient descent
- convex and non-convex optimization
- introducing nonlinearities
- multiple "valleys" with "local minima"



Backpop for multilayer single unit perceptionon



$$E = \frac{1}{2} \sum_{k} (a_k - y_k)^2$$

In words

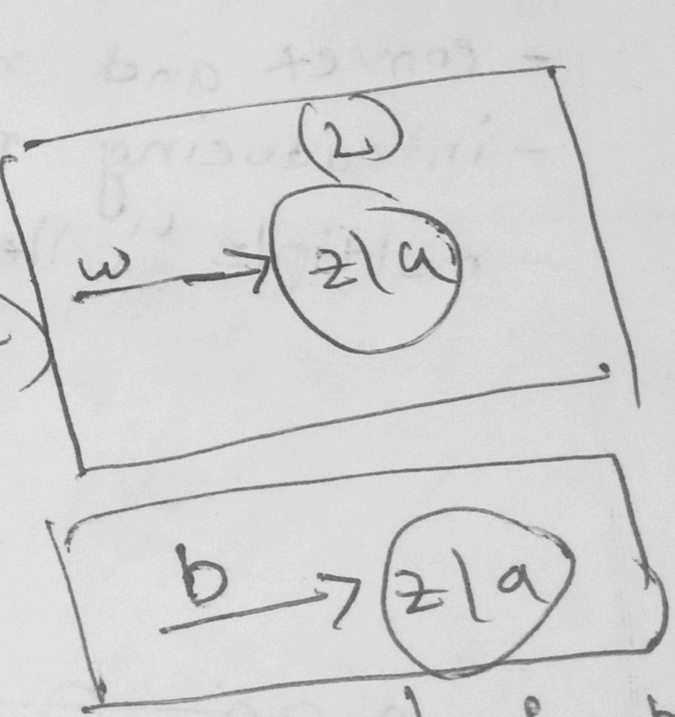
- How does the error E change when we change the activation a by a tiny amount.
- How does a the activation a change when we change the activation 2 by a tiny amount
- How does 2 change when we change the weights w by a tiny amount

Partial derivative

single unit

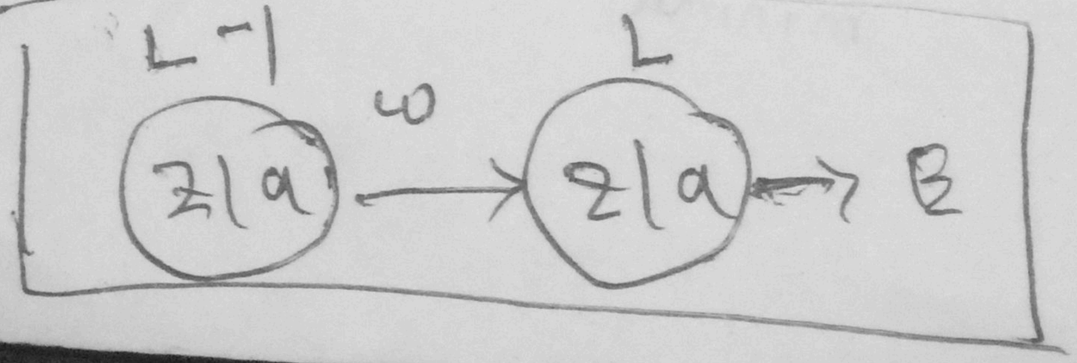
$$\frac{\partial E}{\partial w^{(L)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

$$\frac{\partial E}{\partial b^{(L)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial b^{(L)}}$$



Double unit

$$\frac{\partial E}{\partial w^{(L-1)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \times \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \times \frac{\partial z^{(L-1)}}{\partial z^{(L-1)}}$$



Multilayer Multiunit perceptron

- Modern neural nets (1980's — present) → similar
- shallow networks (not deep / lack of depth) (less hidden layers)
- Feed forward neural networks (Forward propagation)
- Antifieial neural networks (ANN) [brain like structure]
- Fully connected neural networks (FCNN) (All connected)
- FC
- 1980's
- why not deep? → Resource limitation
  → Unavailability of Data

MLP Learning rule

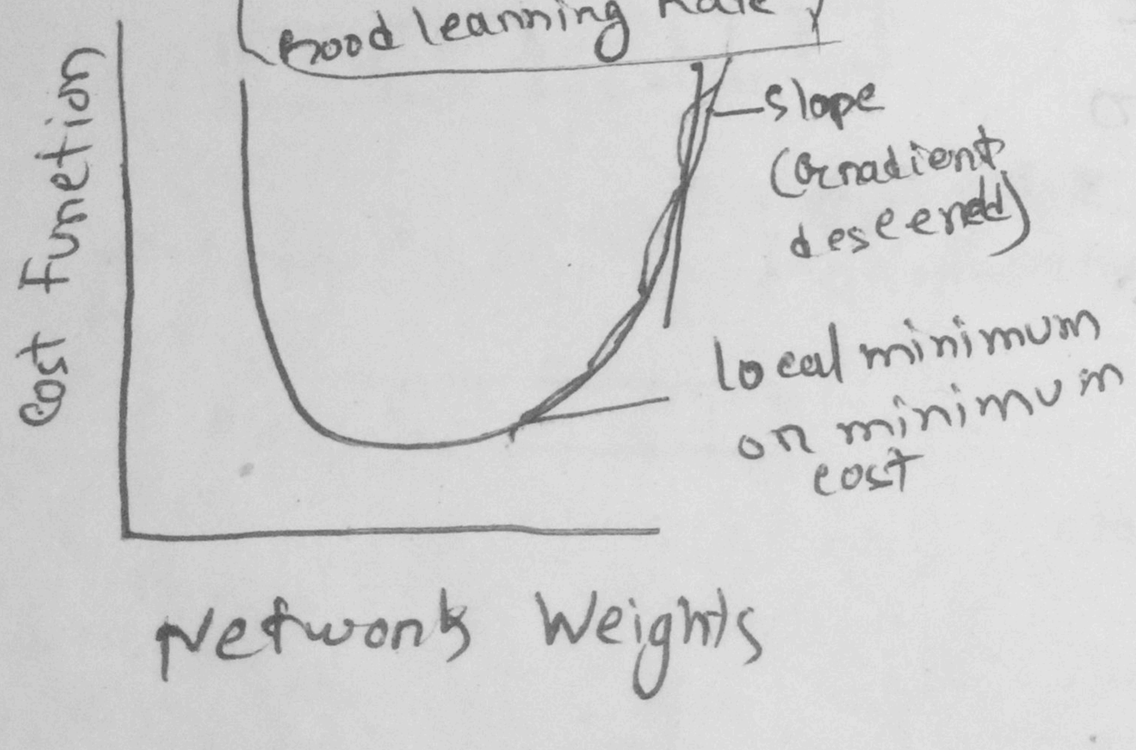
$$w^L_{jk} = w^L_{jk} - \eta \times \frac{\partial E}{\partial w^L_{jk}}$$

η = learning rate

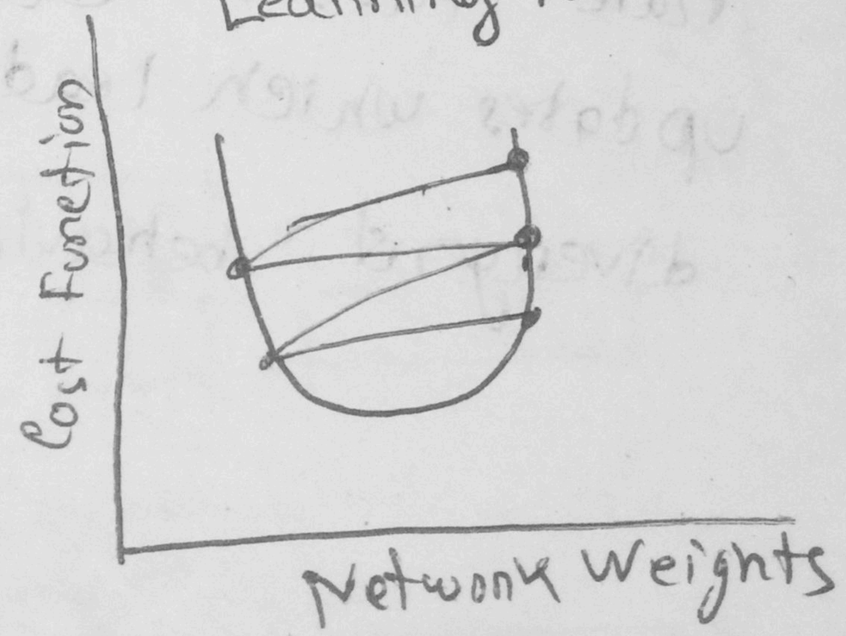
↓ partial derivative

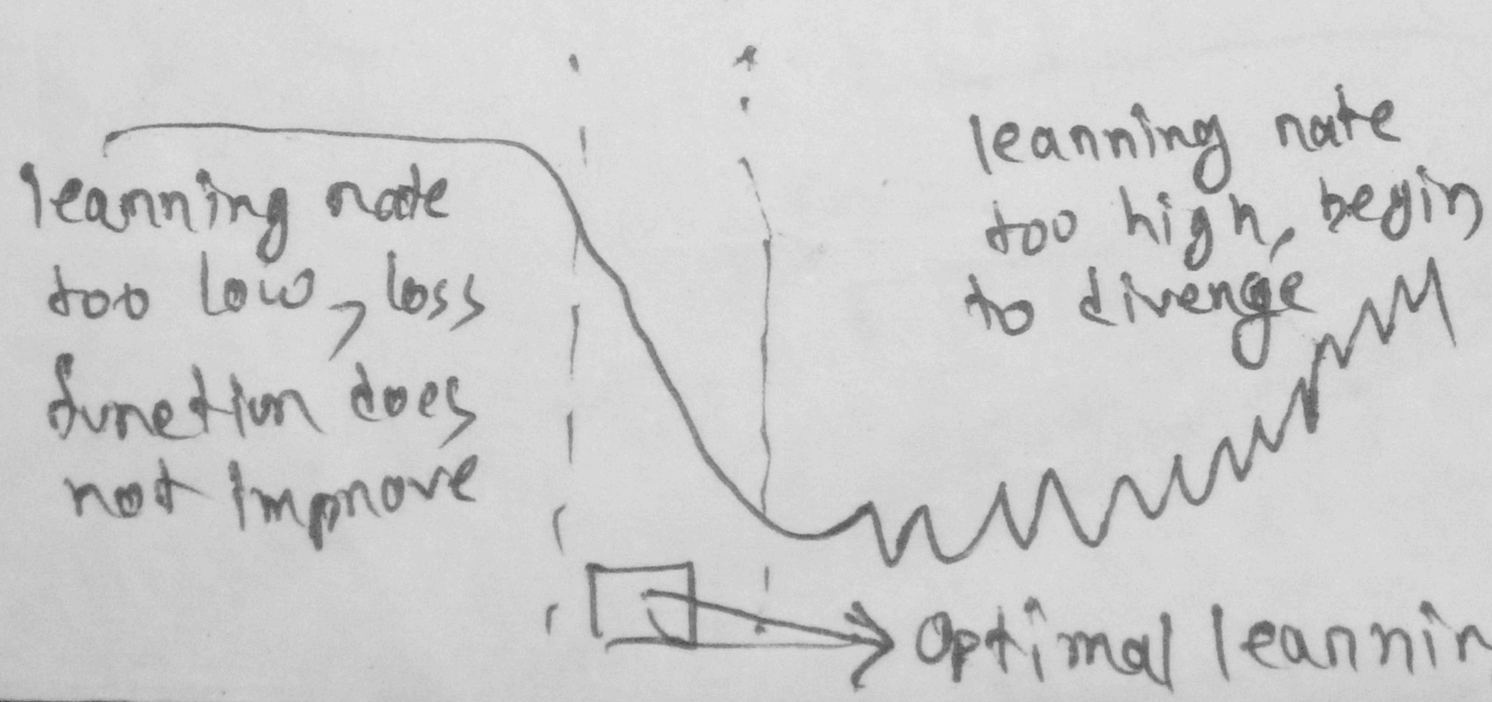
$$b^{(L)} = b^{(L)} - \eta \times \frac{\partial E}{\partial b^{(L)}}$$

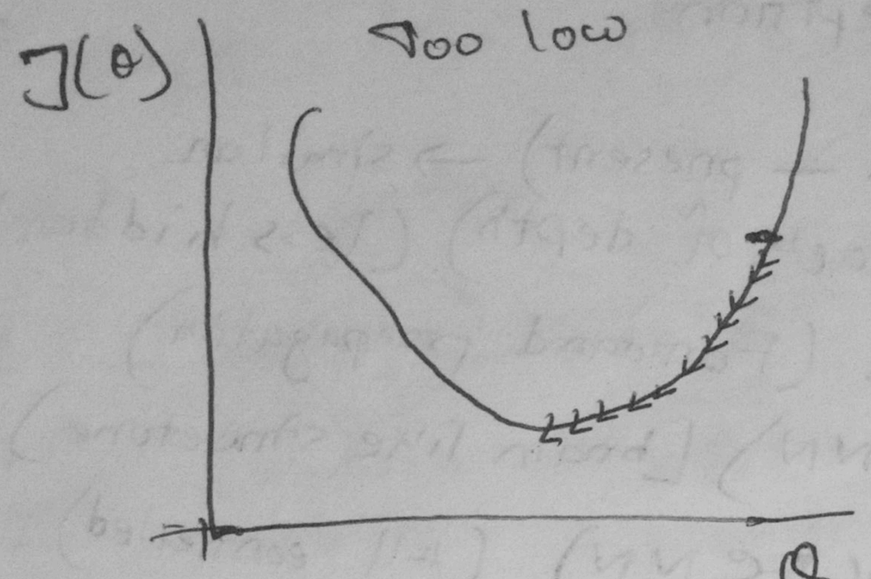
A. Optimal learning / Good learning Rate



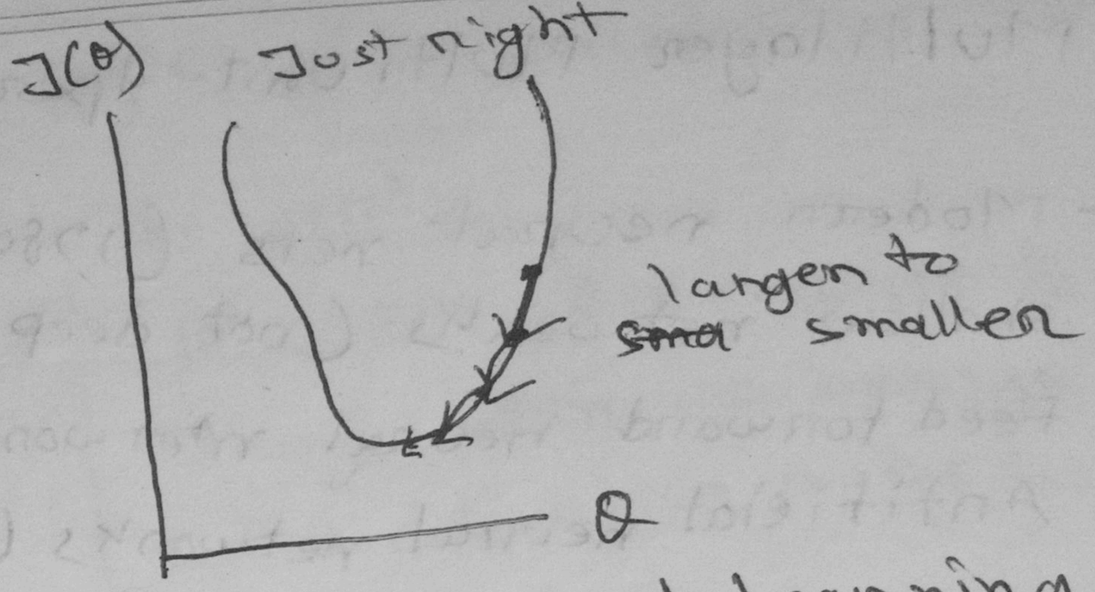
B. Oscillating learning
Learning rate too high



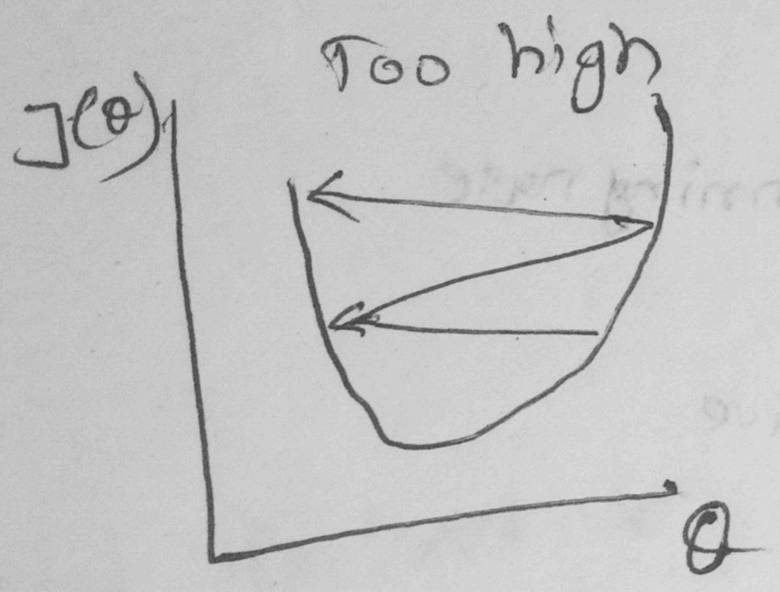




A small leanning nate nequines many updates before neaching the minimum point



The optimal leanning nate swiftly neaches the minimum point.



Too large of a leanning nate causes drastic updates which lead to divergent behaviors.

MP neuron	Perceptron	ADALINE	MLP
1943	1958	1960	1986
Linear aggregation, f	Linear aggregation, f	Linear aggregation, f	Linear aggregation, f
No weight	Weighted sum	Weighted sum	Weighted sum
No bias	No Bias	No bias	Bias
Manual threshold, f	Heaviside step, f	Heaviside step, f	Sigmoid, f
Linear units	Linear units	Linear units	Non-linear units
can not solve XOR	can not solve XOR	sub-optimal solution	can solve
No Learning rule	Simple learning rule	Learning rule on GD(Gradient descend)	Learning rule on BP(Back-propagation
No LR (Learning rate)	LR	LR	LR
Slowest	Faster	~~S~~More Faster	Fastest

Convolution (in 2D)

Polynomials

$2 + 3x + 4x^2 \rightarrow$ 2 Degree

$2 + 3x + 4x^2 + 5x^3 + 6x^4 + 7x^5 \rightarrow$ 5 Degree

$4 + 12x + 25x^2 + 34x^3 + 43x^4 + 54x^5 + 45x^6 + 28x^7 \rightarrow$ 7 Degree

$(k_0 + k_1 x + k_2 x^2)$

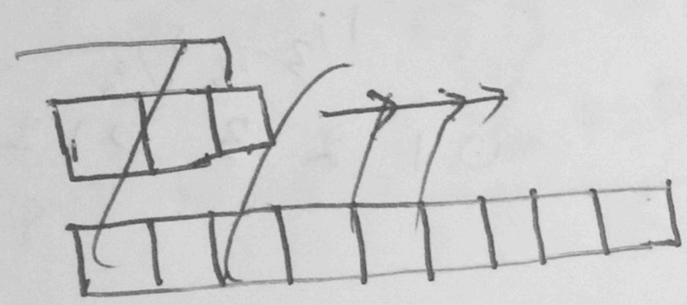
$(a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5)$

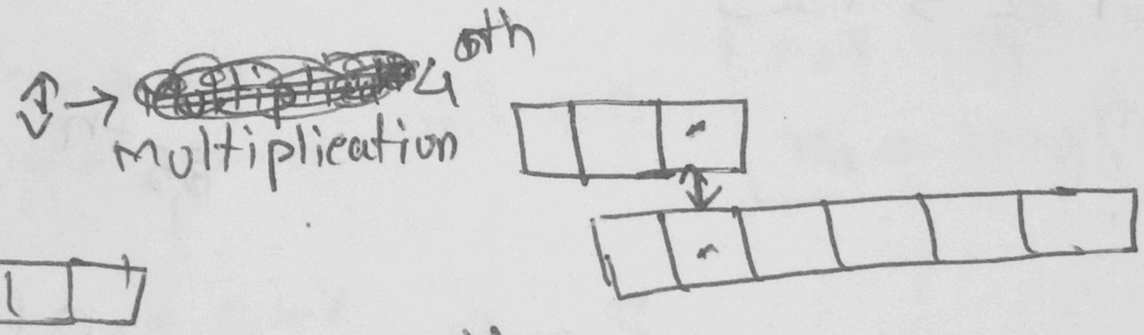
$(k_2 a_2 + k_1 a_3 + k_0 a_4) x^4$

⇒ Multiplying Both ⇒

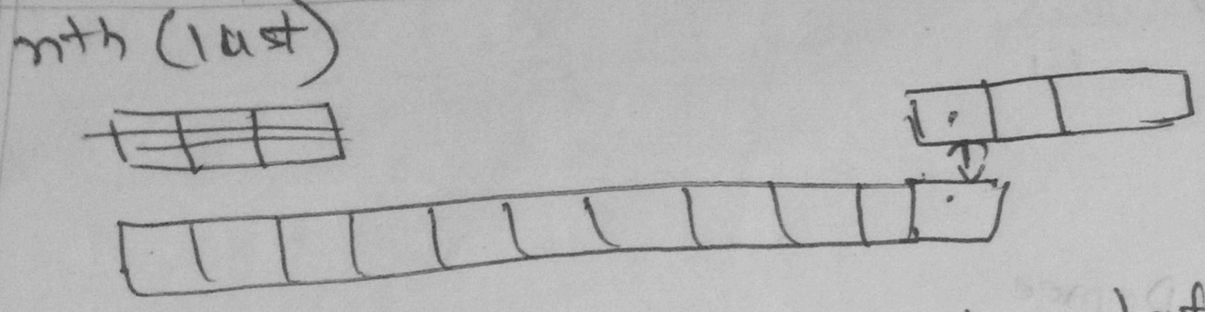
$(K_0 a_0) + (k_1 a_0 + k_0 a_1) x + (k_2 a_0 + k_1 a_1 + k_0 a_2) x^2 + (k_2 a_1 + k_1 a_2 + k_0 a_3) x^3 +$

$(k_2 a_2 + k_1 a_3 + k_0 a_4) x^4 + (k_2 a_3 + k_1 a_4 + k_0 a_5) x^5 + (k_2 a_4 + a_1 a_5) x^6 + (k_2 a_5) x^7$





continuoes



It is convolution simulation with steps

convolution in one dimension

$a = [1/3, 1/3, 1/3]$

$x = [1, 2, 3, 1, 2, 3, 1, 2, 3]$

1st $\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}$

1 2 3 1 2 3 1 2 3 $= 2$

1st

1/3 1/3 1/3

0 1 2 3 0 1 2 0 1 2 0

2nd

$\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}$

1 2 3 1 2 3 1 2 3 $= 2$

2nd

1/3 1/3 1/3

0 1 2 3 0 1 2 3 0 1 2 3 0

7th

$\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3} = 2$

1 2 3 1 2 3 1 2 3

85th

1/3 1/3 1/3

0 1 2 3 0 1 2 3 0 1 2 3 0

---

stride = 1 [shift value]

↳ steps to move

---

stride = 2 [shift value]

↳ steps to move

$$y(1) = \left\{ \begin{matrix} 1/3 & \frac{1}{3} & \frac{1}{3} \\ 0\,0\,1 & 2\,3\,1 & 2\,3\,1\,2\,3\,0\,0 \end{matrix} \right\} = \frac{1}{3}$$

$$y(2) = \left\{ \begin{matrix} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \\ 0\,0 & 1\,2 & 3 & 1\,2\,3 & 1\,2\,3\,0\,0 \end{matrix} \right\} = 1$$

$$y(1) = \left\{ \begin{matrix} & & & & \frac{1}{3}\,\frac{1}{3}\,\frac{1}{3} \\ 0\,0\,1\,2\,3 & 1\,2\,3 & & 1\,2\,3\,0\,0 & \end{matrix} \right\} = 1$$

Total number of

* padding = 2, Extra rightmost/leftmost zero's (counted)
  ↳ we apply padding so that we do not loss any information.

$$\text{Output size} = \frac{n_x + 2P - n_h}{S} + 1 \qquad \begin{bmatrix} P = \text{Padding} \\ S = \text{stride} \end{bmatrix}$$

$$= \frac{9 + 2\cdot 2 - 3}{1} + 1 = 11 \qquad \begin{bmatrix} n_x = \text{length of Input length} \\ n_h = \text{length of filter} \end{bmatrix}$$

$$= \frac{9 + 2\cdot 0 - 3}{1} + 1 = 7$$

$$= \frac{9 + 2\cdot 1 - 3}{2} + 1 = 5$$

Convolution in 2D (Gray scale Image)

Kernel = filter

1	2	3
4	5	6
7	8	9

2D image Gray scale image

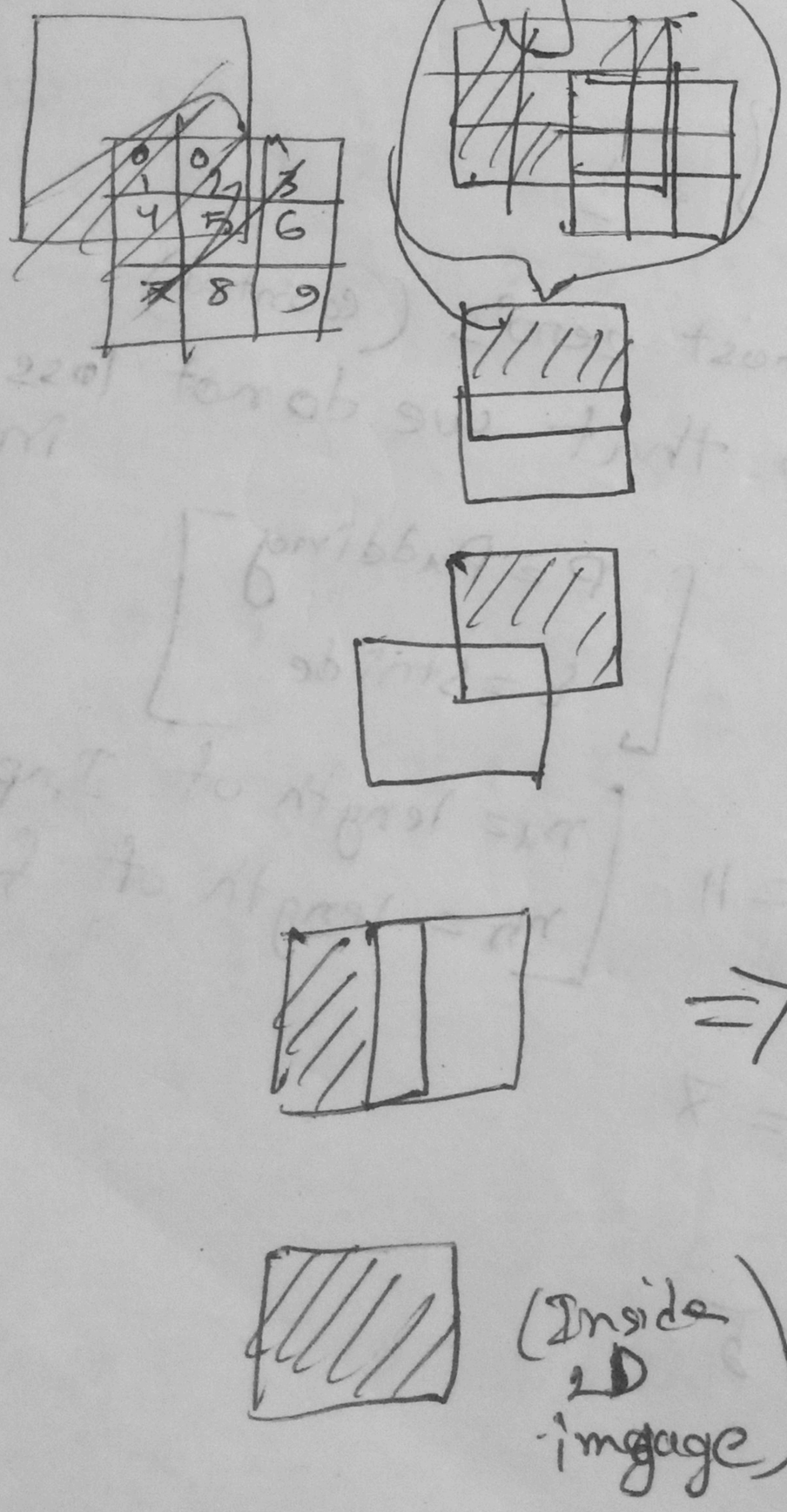
n \ m	0	1	2
0	-1	-2	-1
1	0	0	0
2	1	2	1

Kernel / filter

Kernel size = Matrix shape
↳ normally square shape

→ Average value always in center

. Fill up extra 0's
↳ (Apply padding)



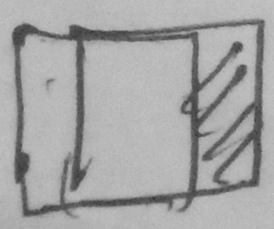
⇒ -13

⇒ -20

⇒ -17

⇒ -18

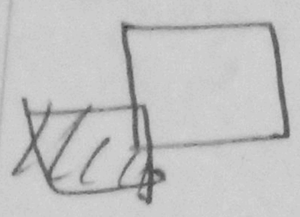
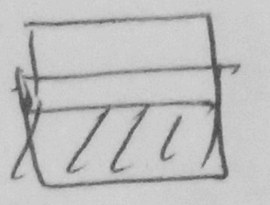
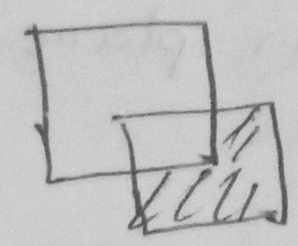
⇒ -24



⇒ -18

→ Kernel

→ Image (2D)
(Applied padding)

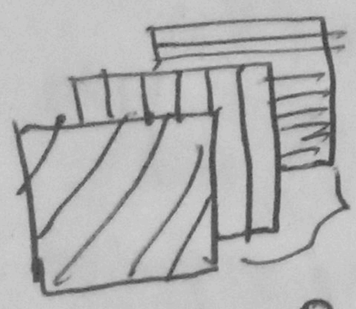
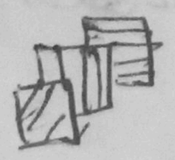




⇒ 13

⇒ 20

⇒ 17

Convolution in 3 dimension

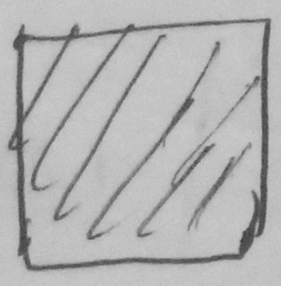
⇒ Filter / Kernel

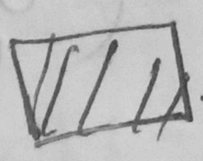
Depth = 3

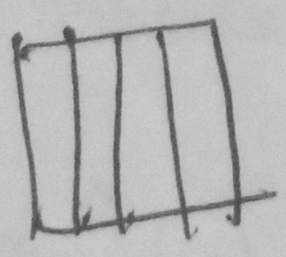
RGB images (Red →
Green →
Blue →)

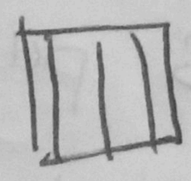
[Their Depth must be same]

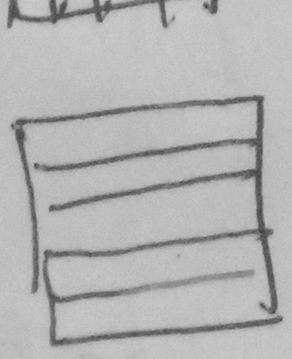
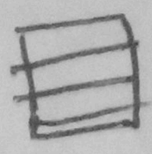
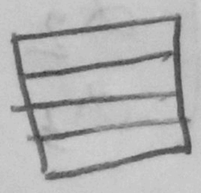
Intermediate Output

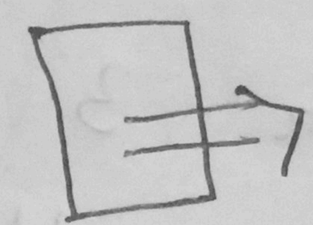










Final output (summation of these output

⇓

Feature map