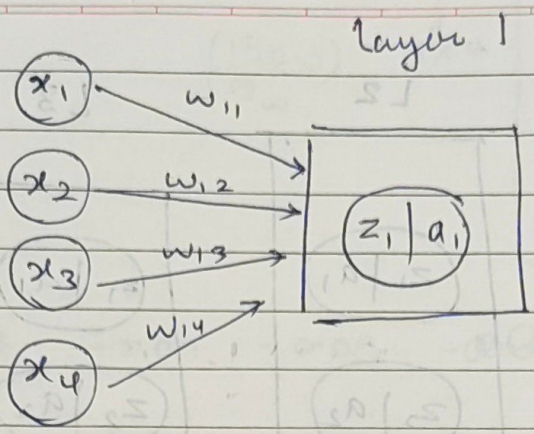
Date 18/07/2025
Page

(1)

x

$$\begin{bmatrix} 0.2 \\ -1.3 \\ 2 \\ 0.01 \end{bmatrix}_{4\times1}$$



$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4$$

$$a_1 = \text{softmax}(z_1)$$

$$\begin{bmatrix} w_{11} \\ w_{12} \\ w_{13} \\ w_{14} \end{bmatrix} = \begin{bmatrix} 0.001 \\ 0.01 \\ -0.005 \\ -1.2 \end{bmatrix}_{4\times1}$$

$$a_1 = \text{softmax}(-0.0348) = 0.$$

1. $z_1 = 0.001 \times 0.2 + 0.01 \times (-1.3) + -0.005 \times (2) + 0.01 \times (-1.2)$

$= 0.0002 - 0.0013 - 0.01 - 0.012$

$= 0.0002 - 0.00225 - 0.01$

$= 0.0026 - 0.01$

$= -0.0123$

$= -0.0348$

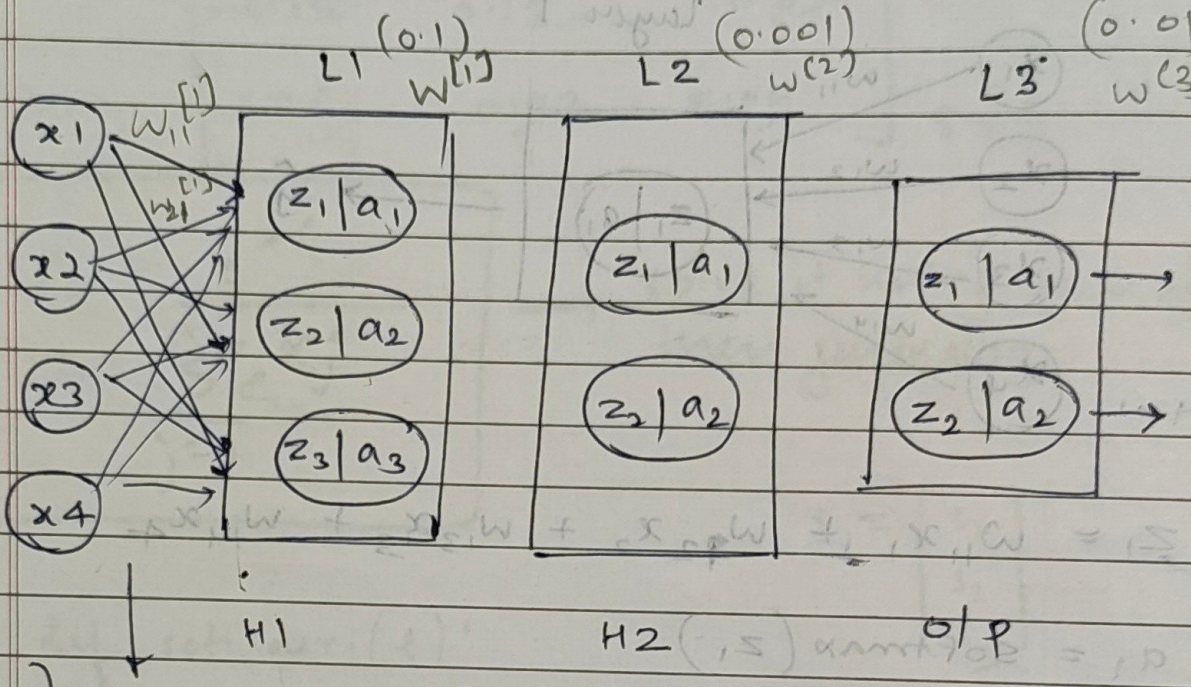
2. $z_1 = X^T W = \begin{bmatrix} 0.2 & -1.3 & 2 & 0.01 \end{bmatrix} \begin{bmatrix} 0.001 \\ 0.01 \\ -0.005 \\ -1.2 \end{bmatrix}$

$= 0.2 \times 0.001 - 1.3 \times 0.01 + 2 \times (0.005) - 1.2 \times 0.01)$

$= -0.0348$

* hidden layer → ReLU
* output layer → softmax (make 2 neurons)

(2)



↑(weights) 12 6 4

ReLU ReLU softmax

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2.4 \\ 1.2 \\ -0.8 \\ 1.1 \end{bmatrix}_{4\times 1}$$

$$\begin{bmatrix} w_{11}^{[1]} \\ w_{21}^{[1]} \\ w_{31}^{[1]} \\ w_{41}^{[1]} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ \vdots \\ 0.1 \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{23} \\ w_{23} & & \\ w_{24} & & \\ w_{31} & & \\ w_{32} & & \\ w_{33} & & \\ w_{34} & & \end{bmatrix} = \begin{bmatrix} 0.1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0.1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & \cdots \\ w_{12} & w_{22} & \cdots \\ w_{13} & & \\ w_{14} & & \end{bmatrix}^{[1]} = \begin{bmatrix} 1 & 2 & 3 \\ \vdots & & \\ 3 & & \end{bmatrix}$$

$$z_1 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix} \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{23} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix} \begin{matrix} a_1 & a_2 & a_3 \end{matrix}$$

no. of weights from next activation function) & n(input).

$$= \begin{bmatrix} -2.4 & 1.2 & -0.8 & 1.1 \end{bmatrix}_{1\times 4} \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}_{4\times 3}$$

$$t_1 = \begin{bmatrix} -0.09 & -0.09 & -0.09 & -0.09 \end{bmatrix}$$

$$t_2 = \begin{bmatrix} -0.09 & -0.09 & -0.09 & -0.09 \end{bmatrix}$$

$$o/p = \begin{bmatrix} a_1(-0.09) & a_2(-0.09) & a_3(-0.09) \end{bmatrix}$$

$$o/p = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}_{1\times 3} \begin{bmatrix} 0.001 & 0.001 \\ 0.001 & 0.001 \\ 0.001 & 0.001 \end{bmatrix}_{3\times 2}$$

$$= \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} a_1(0) & a_2(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$L_3 = \begin{bmatrix} 0 & 0 \end{bmatrix}_{1\times 2} \begin{bmatrix} 0.01 & 0.01 \\ 0.01 & 0.01 \end{bmatrix}_{2\times 2}$$

$$L_3 = \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} \text{softmax}(0) & \text{softmax}(0) \end{bmatrix}$$

$$o/p = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

Assignment 1 - Deep learning - Santhosh

Date
Page

(1.) Output of last FC-layer = $\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$

input → LI - - - - - last layer

○ → 2.0

○ → 1.0

○ → 0.1

To convert this net into predicting class prob. for 3 classes, we can use a softmax classifier.

Softmax classifier takes an input vector i.e. logits & returns an output vector of prob. for each class in the input vector.

Sum of this output vector = (1)

$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ ∴ $\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$ → Softmax → $\begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$

$\bar{x}$ — last layer — o/p prob. vector

$$\text{Softmax} = P(y=i) = \frac{e(x_i)}{\sum e(x_j)}$$

prob. of ith class in vector (input)

$$\therefore P(y=x_1) = \frac{e(2.0)}{e(2.0)+e(1.0)+e(0.1)} = 0.658$$

→

$$P(y=x_2) = \frac{e(1.0)}{11.204} = \frac{2.71}{11.204} = 0.2418$$

$$P(y=x_3) = \frac{e(0.1)}{11.204} = \frac{1.105}{11.204} = 0.0986$$

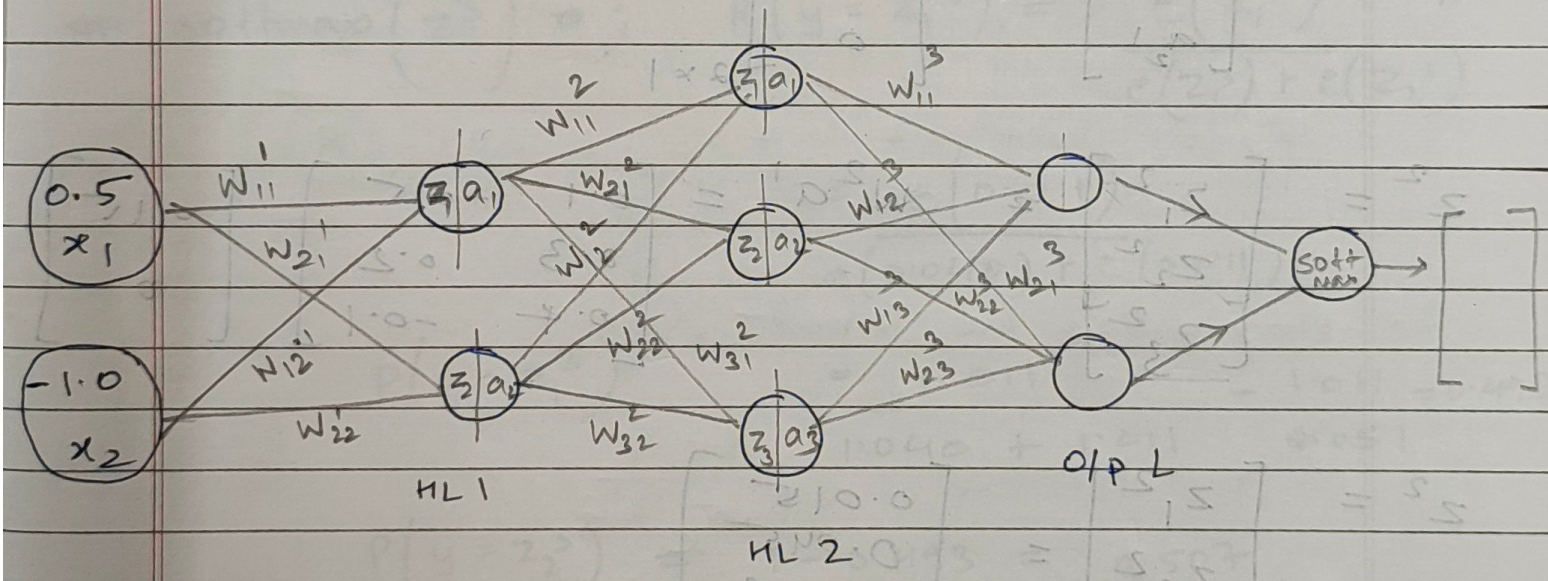
∴ Output vector i.e. probability vector for 3 classes is

$$\begin{bmatrix} 0.658 & 0.2418 & 0.0986 \end{bmatrix}$$

(2.) Given: $X = [0.5 \quad -1.0]$, 2 hidden layers with 2 & 3 neurons

o/p layer = 2 neurons

ReLU in hidden layers / softmax at o/p layers



Input $X = \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix}_{2\times 1}$ weight (layer 1) $W^1 = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}$

Using uniform distribution, for random initialization $= \begin{bmatrix} 0.1 & -0.1 \\ 0.2 & 0.3 \end{bmatrix}_{2\times 2}$

$$\therefore \begin{bmatrix} 0.1 & -0.1 \\ 0.2 & 0.3 \end{bmatrix}$$

Now, $Z^1 = W^1 X = \begin{bmatrix} 0.1 & -0.1 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.15 \\ -0.2 \end{bmatrix}$

$$z^1 = \begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix} = \begin{bmatrix} 0.15 \\ -0.2 \end{bmatrix}$$

$$a^1 = ReLU(z^1) = \begin{bmatrix} ReLU(z_1^1) \\ ReLU(z_2^1) \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0 \end{bmatrix} = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$$

layer 2 initialize weights

$$W^2 = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}_{3\times 2} = \begin{bmatrix} 0.1 & -0.5 \\ 0.3 & 0.2 \\ 0.4 & -0.1 \end{bmatrix}_{3\times 2}$$

$$a^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0 \end{bmatrix}_{2\times 1}$$

$$z^2 = \begin{bmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{bmatrix} = W^2 a^1 = \begin{bmatrix} 0.1 & -0.5 \\ 0.3 & 0.2 \\ 0.4 & -0.1 \end{bmatrix} \begin{bmatrix} 0.15 \\ 0 \end{bmatrix}$$

$$z^2 = \begin{bmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{bmatrix} = \begin{bmatrix} 0.015 \\ 0.045 \\ 0.006 \end{bmatrix}$$

$$a^2 = ReLU(z^2) = \begin{bmatrix} 0.015 \\ 0.045 \\ 0.006 \end{bmatrix}$$

output layer

$$W^3 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & -0.1 \\ 0.4 & 0.7 & 0.3 \end{bmatrix}$$

$$z^3 = \begin{bmatrix} z_1^3 \\ z_2^3 \end{bmatrix} = W^3 \cdot a^2 = \underset{2\times 3}{\begin{bmatrix} 0.1 & 0.2 & 0.1 \\ 0.4 & 0.7 & 0.3 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} 0.015 \\ 0.045 \\ 0.006 \end{bmatrix}}$$

$$z^3 = \begin{bmatrix} 0.011 \\ 0.040 \end{bmatrix}$$

\# softmax (z^3) ; $P(y = z_1^3) = \dfrac{e(z_1^3)}{e(z_1^3) + e(z_2^3)}$

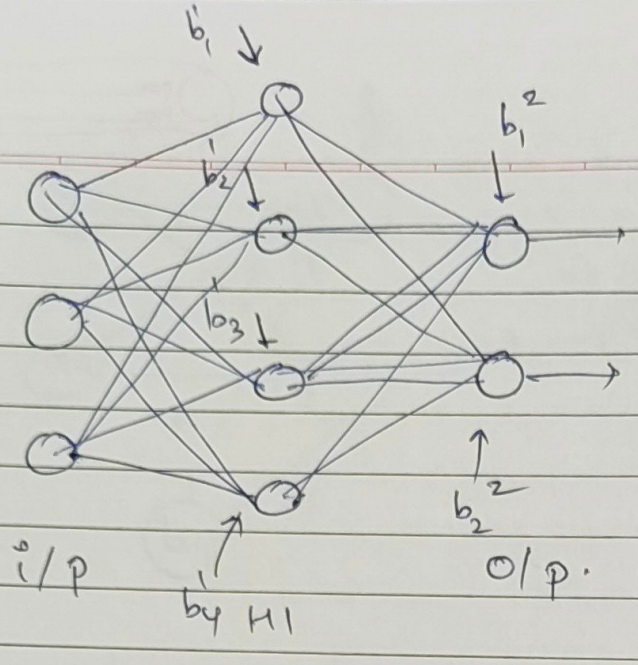
$$= \frac{e(0.011)}{e(0.040) + e(0.011)}$$

$$P(y = z_1^3) = \frac{1.011}{1.040 + 1.011} = \frac{1.011}{2.051} = 0.493$$

$$P(y = z_2^3) = 1 - 0.493 = 0.507$$

∴ output vector $= \begin{bmatrix} 0.493 \\ 0.507 \end{bmatrix}$

④



Total no. of parameters.

$w^1 = 4 \times 3 = 12$

$b^1 = 4$

$w^2 = 4 \times 2 = 8$

$b^2 = 2$

$\therefore$ total = (26)

⑥ $x = [2, 4, 6, 8, 10]$ $\quad \gamma = 2 \quad | \quad \beta = -1$

Batch Normalization.

1. Calculate mean & variance for sample set.

$$\mu = \frac{2+4+6+8+10}{5} = \frac{30}{5} = 6.$$

$$\sigma = \sqrt{\sum_{i=1}^{5} \frac{(x_i - \mu)^2}{5}}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

$$\sigma = \sqrt{\frac{(2-6)^2 + (4-6)^2 + (6-6)^2 + (8-6)^2 + (10-6)^2}{5}}$$

$$\sigma = \sqrt{\frac{16+4+4+16}{5}} = \sqrt{8} = 2\sqrt{3} = 3.464$$

$$\therefore \boxed{\mu = 6 \quad | \quad \sigma = 3.464.}$$

2. Normalize the sample vector. X

$$\boxed{\bar{x} = \frac{x-\mu}{\sigma}} \quad \therefore \bar{x}_1 = \frac{x_1 - \mu}{\sigma} = \frac{2-6}{3.464}$$

$$\bar{x}_1 = -1.154$$

$$x_2 = \frac{4-6}{3.464} = \frac{-2}{3.464} = -0.577$$

$$x_3 = 0 \quad , \quad x_4 = +0.577 \quad , \quad x_5 = +1.154.$$

$$\bar{X} = [-1.154, -0.577, 0, +0.577, +1.154]$$

3. Scale & shift the output.

$$y = \gamma \bar{x} + \beta$$

$$y = 2\bar{x} - 1$$

$$y = 2[-1.154 \quad -0.577 \quad 0 \quad +0.577 \quad 1.154] - 1$$

$$= [-2.308 \quad -1.154 \quad 0 \quad 1.154 \quad 2.308] - 1$$

$$y = [-3.308 \quad -2.154 \quad -1 \quad 0.154 \quad 1.308]$$

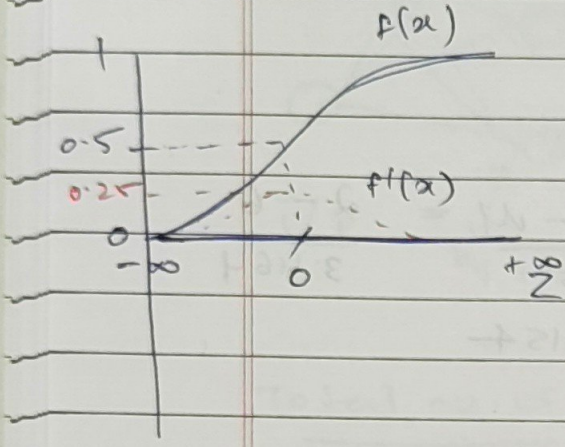
$\gamma = 2$

$\beta = -1$

(5) Gradient Vanishing problem.

i) Sigmoid.

- output value $R \in [0, 1]$
- Derivative $f'(x) = f(x)(1 - f(x))$
  - ↳ max-derivative = 0.25
- As the input becomes large → ∞, ~~then~~ the output value saturates at 1 or 0
  - → As the outputs are saturated at 1, 0, the derivative becomes 0
  - → ~~During~~ ~~backprop~~ This leads to vanishing gradient problem.
  - → Gradient shrinks as the backpropagation involves chain rule multiplication
  - → Training stagnates, no learning occurs.



Q.7. ~~Role of Dropout~~

Role of 'keep probability' 'p' in dropout.

- dropout randomly removes / deactivates bunch of neurons during training to prevent overfitting.
- keep probability (p) is a hyperparameter which decides the number of neurons to be kept active during training.
- dropout using keep-prob is a regularization technique
  i) low 'p' value signifies higher regularization & vice-versa.
- it causes randomization, so no memorization of the training data.

To keep the activation ~~same~~ magnitude same during training & testing stage, we scale the activation by a factor of keep probability.

It is done in two ways ~~1) stan~~

1) Standard dropout.
   - In inference, there is no dropout occuring. ∴, to ensure ~~uniform~~ consistent activation magnitude at test time, the activations are scaled up by multiplying with keep prob.
   - This scales down the activation, at testing

2) Inverted dropout (up)
   ↳ here, the activations are scaled at training time itself by ~~dividing~~ them with multiplying $\left(\frac{1}{p}\right)$
   ↳ no scaling at test time, as it's already accounted for at training time.

(8) input image -

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

X

filter -

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

3 x 3

output → 4 x 4.

marks the vertical edge between 10's region & 0's region.

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4.

using the given convolution filter helps in detecting vertical edges present in the sample image between 10 & 0 pixel values.

(9) The Impact of increasing the no. of convolutional layers in a CNN include :-

1) vanishing gradient problem.
   ↳ As the number of layers increasing ↑'s & if sigmoid is used as an activation function; then during the backpropagation due to chain multiplication of derivatives the gradient gets pushed to very small value.
   ↳ this affects the training process.

2) With deeper layers, the number of parameters ↑ significantly, leading to high computational cost & memory

3) With increasing no. of layers in the CNN, the original input image is lost due to series of conv. operation. At deeper layers, the net starts to learn from the noise & represent -ation.

4) Higher risk of overfitting on a small dataset.

(10)

(10) Weight sharing

- same filter i.e. set of weights is applied across the entire input image.
- It reduces the no. of parameters as compared to the fully connected nets.
- makes the model translation - invariant
  ↳ feature can be detected anywhere in the sample input.

Sparsity of connection

- Each neuron in the conv. layer is connected only to a small local region of the input (receptive field), not to every input pixel.
- leads to fewer computations & parameters, making CNN efficient

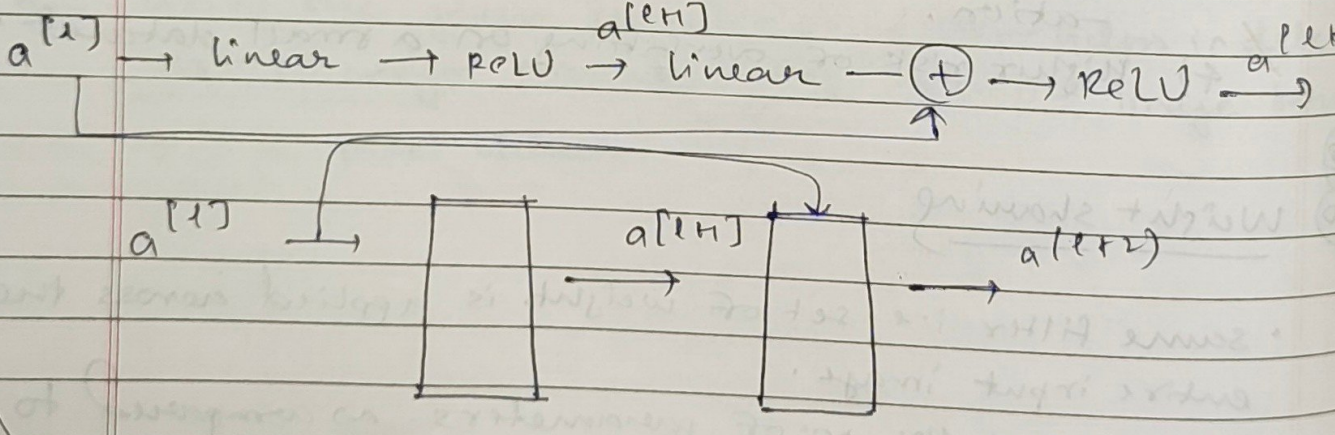
(11) ResNet:

- it introduces skip (residual) connections → instead of each layer learning $H(x)$, it learns a residual function $F(x) = H(x) - x$.
- output of Residual block: $y = f(x) + x$.

$$H(x) = f(x) + x$$

letting the input pass as it is

if the function is just the identity mapping i.e. $(H(x) = x)$, the residual becomes $F(x) = 0$. which is easy to learn.



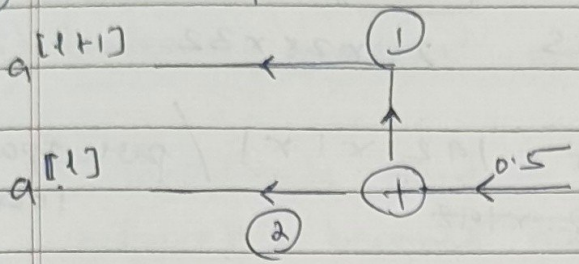
when $w^{[l]}$ becomes very small then,

$$a^{[l+2]} = g\left[w^{[l+2]} a^{[l+1]} + a^{[l]}\right]$$

becomes small

↳ becomes an identity map, as it learns from the previous representation.

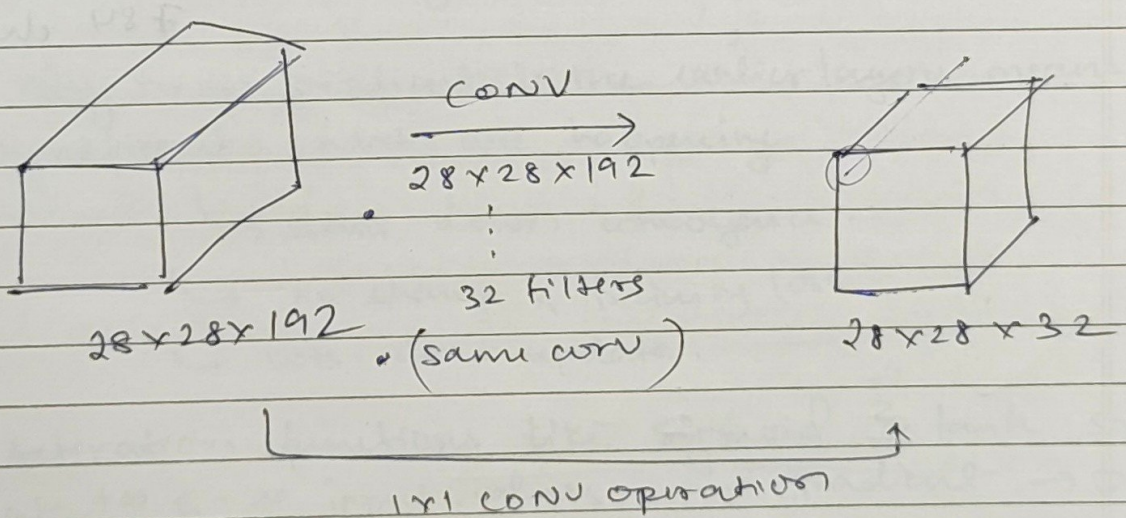
(12) How skip connection improve gradient flow in Res Net



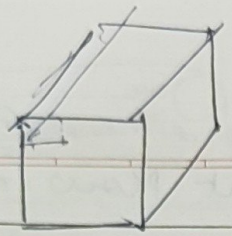
- Skip connections creates shortcuts for gradient during backprop.
- In path ①, the standard flow of gradient from the lth layer to initial layers, the gradient might diminish when it reaches initial layers.
- In path ②, the skip connection allows the flow of gradient directly to the initial layers.

$$\therefore \frac{\partial L}{\partial W} = \frac{\partial L}{\partial W}\overset{①}{\text{(diminish)}} + \frac{\partial L}{\partial W}\overset{②}{}$$

(13)



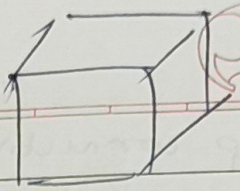
Size of filter = 28 x 28 x 192 = 150,528 wts.
Total points = 28 x 28 = 784.
1 filter = 150,528 x 784
∴ 32 filters = 32 x 150,528 x 784
≈ 3.78×10^9 multiplications.



28 × 28 × 192

1 × 1 CONV
× 192

32 filters



28 × 28 × 32

no. of weights in 1 filter = 192 × 1 × 1 / per spatial location

~~32 filters = 32 × 192~~

∴ Total locations = 28 × 28 = 784

∴ Total multiplication / filter = 784 × 192
= 150,528

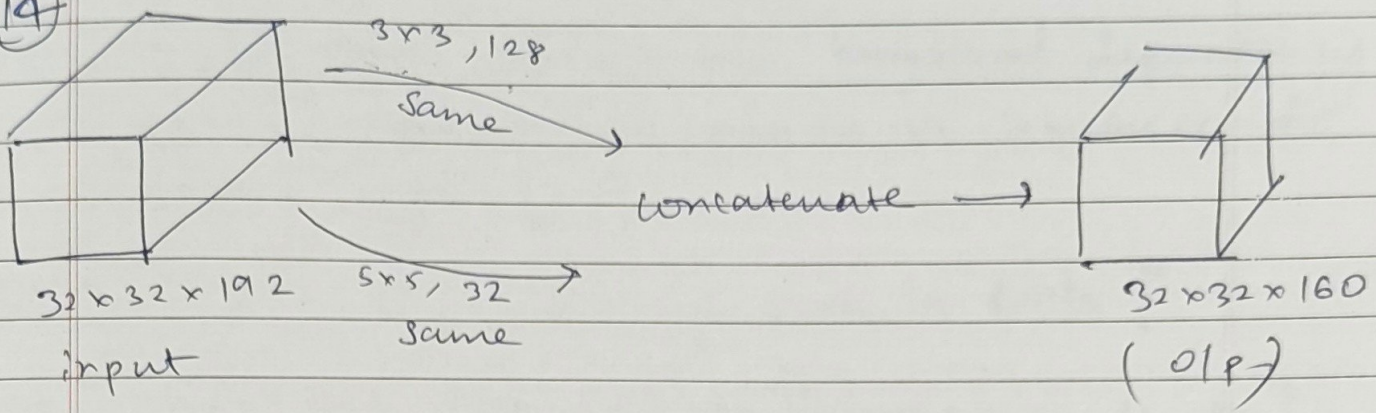
32 filters = 32 × 150,528 = 4,816,896.

≈ 4.8 mil ~~parameter~~ multiplications

Saving factor = $\frac{3.78 \times 10^9}{4.82 \times 10^6} \approx 784$

1 × 1 conv is 784 cheaper

(14)



a) input = 32×32×192
output = 32×32×160
128 filters of 3×3
32 filters of 5×5

Multiplications for 3×3 CONV filters.

Filter size = 3×3×192 = 1728 mults / position.
Spatial positions = 32×32 = 1024
Filters = 128.
Total = 128×1024×1728 = 226,492,416.

# multiplications for 5×5
Total = 157,286,400.

Total of 3×3 & 5×5 = 383,778,816.

b)