1D2

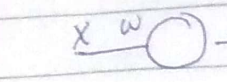

```
1 import tensorflow as tf
2 import numpy as np
3 import tensorflow import keras
4 model = tf.keras.Sequential ([keras.layers.Dense(units=1, input_
                                shape=[1])
5 model.compile(optimizer='sgd', loss='mean_squared_error')
6 model.fit(x, y, epochs=10)
7 print(model.predict([]))
```

4 & 5 model - structure of entire neural network
Sequential - all ~~neurons~~ layers are connected in sequence.
dense - all neurons are interconnected
units=1 - no. of neurons
input_shape - type of I/p going into neuron
6. This stmt runs the model to find the function
epochs - no. of iterations.

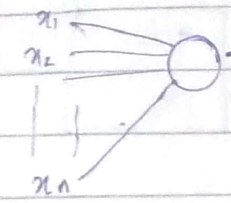
As of now we have not given I/p data & prediction
Intended func: y=x data. After line 5, add:

```
x = np.array([1.0, 2.0, 3.0, 4.0, 6.0], dtype="float")
y = np.array([1.0, 2.0, 3.0, 4.0, 6.0], dtype="float")
```

& in line 7 inside the predict give a value say 5.0



Need only 1 weight parameter.



Need n weight parameters.

$n^{[l]}$ - no of nodes in layer l.
$w^{[l]}$ - wt. parameters in layer l.
$b^{[l]}$ - bias in layer l.

Gradiant descent = For all i, the equ. below is repeated until $\frac{\delta J}{\delta w_j} = 0$

learning rate (> 0) alpha

cost

$$w_i = w_i - \alpha \frac{\delta J}{\delta w_i}$$

any of them

Repeat until we can't change $\frac{\delta J}{\delta w_i}$

for i in weights → set of all wts irrespective of all layers

$$w_i = w_i - \alpha \frac{\delta J}{\delta w_i}$$

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)})$$ → basically avg. loss over all training examples

Super script in paranthesis means we are talking about i^{th} example

loss ← $\mathcal{L} = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$. Cost func. becomes loss func. as soon as we avg

We don't use methods of finding maxima / minima in calculus $\left(\frac{dy}{dx}, \frac{dy^2}{dx^2}\right)$ here as it will become NP-hard in multi dimensions while gradient descent is a light wt algorithm.

Here we are trying to minimise cost function &

degree of
this will help us tell missed classifications
($\hat{y} \in (0,1) \Rightarrow$ If $\hat{y} \geq 0.5$, $\hat{y} = 1$ else 0)

If actual $y = 1$ & our pred says $\hat{y} = 0$
$\log 0 = -\infty$ $\log 1 = 0$
$\mathcal{L} = -(-\infty + 0) = \infty$ i.e we have a huge penalty.

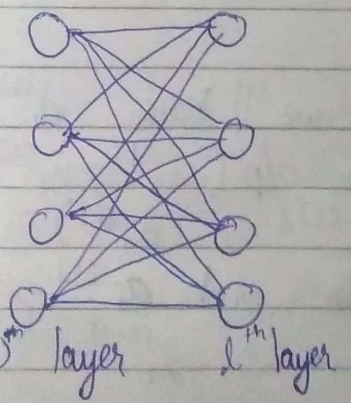
$\hat{y}$ comes from $\sigma(z) = \frac{1}{e^{-z}+1}$ so $\hat{y}$ can be 0 only if $z = -\infty$.

Now z can't be $\neq -\infty$, so $\hat{y} \neq 0$.
If $\hat{y} = 0.0001$, then loss will be very ↑.
If $y = \hat{y}$, then loss will be 0 i.e no penalty.

So we want to minimise $\mathcal{L}$ & J as it will make our classifier better

In order to minimise J, gradient descent will help as we find $\frac{\delta J}{\delta w_{ij}^{[l]}}$'s minimum value (using basic calculus)

Dimensions of $w^{[l]}$ in terms of $n^{[l]} : n^{[l]} \times n^{[l-1]}$



To write previous eq. more accurately,

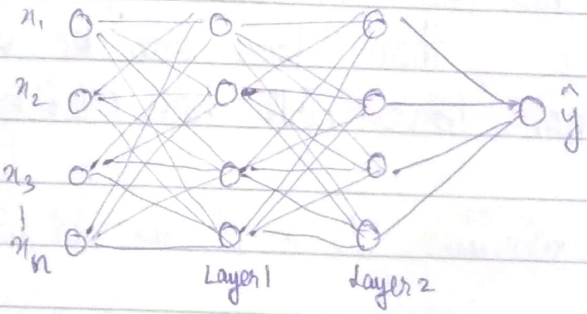
$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \frac{\delta J}{\delta w_{ij}^{[l]}}$$

By convention, we don't count I/p layer as layers!

This will contain 4 loops:

Repeat until there is no updation:
  For all layers l = 1 to L → last layer
    For all i = 1 to $n^{[l]}$
      For j = 1 to $n^{[l-1]}$

$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \frac{\delta J}{\delta w_{ij}^{[l]}}$$



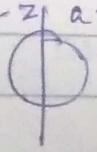
$W^{[2]} = 4 \times 4$

$W_1[1] = 4 \times n$

L is a function of $y, \hat{y}$, $\hat{y}$ is a function of z, z is a function w which is a func. of a's which is a func. of previous layer's w's & a's & soon

Linear part ← z | a → Non linear part O/p

O/p



Neuron

In a layer l, we'll have $n^{[l]}$ neurons, so we will have $n^{[l]}$ linear O/p ($z_1^{[l]}, z_2^{[l]} \dots z_n^{[l]}$) & $n^{[l]}$ non linear O/p ($a_1^{[l]}, a_2^{[l]} \dots a_n^{[l]}$)

How to compute each $a_i^{[l]}$? It will depend on O/p from previous layer. i.e $a_1^{[l-1]} \dots a_{n^{[l-1]}}^{[l-1]}$

This whole thing is 1 no ← $z_1^{[l]} = W_{11}^{[l]} a_1^{[l-1]} + W_{12}^{[l]} a_2^{[l-1]} + \dots W_{1n^{[l-1]}}^{[l-1]} a_{n^{[l-1]}}^{[l-1]}$

Linear O/p of only 1st neuron.

$$= \sum_{j=1}^{n^{[l-1]}} W_{1j} a_j^{[l-1]}$$

These are the forward passes.

Every layer in a NN computes these 2 eqn

$$a^{[l]} = \frac{1}{1+e^{-z^{[l]}}}$$

$$z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g(z^{[l]})$$

→ any activation

} Capitalised notation for matrices

Dimensions:

$w - n^{[l]} \times n^{[l-1]}$

$A^{[l-1]} - n^{[l-1]} \times 1$

$b - n^{[l]} \times 1$

$\Rightarrow z^{[l]} - n^{[l]} \times 1$

$A^{[l]} = dim(z^{[l]}) = n^{[l]} \times 1$

So this will help us get rid of some of the for loops

So in GD, for all minima & maxima, $\frac{\delta I}{\delta w_{ij}^{[l]}} = 0$

If minima, $\forall i, j, l \quad \frac{\delta I}{\delta w_{ij}^{[l]}} = 0$ (not true vice-versa)

→ GD doesn't give guaranteed minimum, it does bcoz minimum work.

For our NN, our parameters are $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \ldots w^{[L]}, b^{[L]}$.

If we have 3 layers, where there is I/p, 6 neurons in 1st layer, 3 neurons in 2nd layer & 1 in last layer,

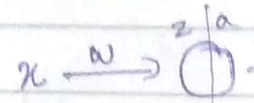
no. of parameters $= ((5 \times 6) + 6) + ((6 \times 3) + 3) + ((3 \times 1) + 1)$

$= 61$ parameters

So once our GD completes, we have found best possible value of parameters for given initialisations.

How to find $\frac{\delta I}{\delta w_{ij}}$?

Suppose we have only 1 neuron in 1 layer:



(no bias)

$z = wx \quad \hat{y} = \frac{1}{1+e^{-z}}$

GD eq: $w = w - \alpha \frac{dL}{dw}$ → as there is only 1 layer.

$\frac{dL}{dw} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dz} \frac{dz}{dw}$ } Use this and it comes from dependencies & chain rule

Suppose we still have 1 neuron, but n I/p:

$z = \sum_{i=1}^{n} x_i w_i + b$

$\hat{y} = a = \frac{1}{1+e^{-z}}$

In prvs 4 loops eq.: L & i loop disappear

For $j = 1$ to $n \rightarrow$ no. of I/p as 1 layer

$w_{ij} = w_{ij} - \alpha \frac{\delta L}{\delta w_{ij}}$ → So we have n equations of this & one equation for bias.

Repeat

for $l = 1$ to L

for $i = 1$ to $n^{[l]}$

for $j = 1$ to $n^{[l-1]}$

$w_{ij} = w_{ij} - \alpha \frac{\delta I}{\delta w_{ij}^{[l]}}$

$b_i^{[l]} = b_i^{[l]} - \alpha \frac{\delta I}{\delta b_i^{[l]}}$

Just cause we are going over all w, doesn't mean it is a global minima.

activation func. at layer L

$$\hat{y} = a^{[L]} = g^{[L]}(z^{[L]})$$

$$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$$

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y^{[L]}, \hat{y}^{[L]})$$

So J is a func of $\hat{y}$ which in turn is a func. of $g^{[L]}$ & $z^{[L]}$ which is a func. of $W^{[L]}, a^{[L-1]}, b^{[L]}$

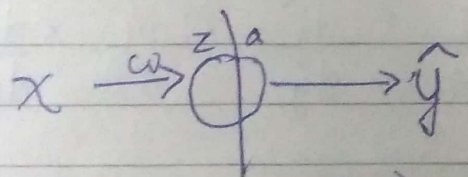
Out of these, $g^{[L]}$, $W^{[L]}$, $b^{[L]}$ are final parameters that don't depend on previous values.

$$a^{[L-1]} = g^{[L-1]}(z^{[L-1]})$$
$$z^{[L-1]} = W^{[L-1]} A^{[L-2]} + b^{[L-1]}$$

→ These again don't depend on prvs values

So in the end J depends on $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots W^{[L]}, b^{[L]}$

To find $\frac{\partial J}{\partial W_{ij}^{[L]}}$ we use back propagation.



$$\hat{y} = a = g(z) \quad (1)$$
$$z = wx \quad (2)$$

In prvs code, first 3 loops go away & our code becomes:

repeat

$$W = W - \alpha \frac{\partial L}{\partial W}$$

We are using 1 training example, so J becomes L (as $m = 1$)

How to compute $\frac{\delta L}{\delta w}$?

If $y = e^{x^2+2}$

$$\frac{dy}{dx} = e^{x^2+2}(2x)$$

Now if we write this as:

$$y = e^z$$

$$z = x^2 + 2$$

$$\frac{dy}{dx} = \frac{dy}{dz} \times \frac{dz}{dx}$$

$$= e^z \times 2x = 2x e^z$$

So $\frac{dL}{dw} = ?$

L is a direct func. of y & $\hat{y}$ but y is not a parameter (ie we have no control over y, it's given)

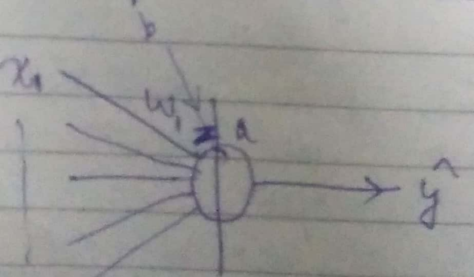
$$\frac{dL}{dw} = \frac{dL}{d\hat{y}} \times \frac{d\hat{y}}{dz} \quad \text{(Using ①)}$$

(Back propagation) $$\frac{dL}{dw} = \frac{dL}{d\hat{y}} \times \frac{d\hat{y}}{dz} \times \frac{dz}{dw} \quad \text{(Using ②)}$$

If we had m training examples, there'd be no change except L would become J

Next level of complication would be:

(logistic regression)



$$\hat{y} = a = g(z)$$

(Say g is sigmoid)

In prvs code, 1st for loop goes away as $L=1$, 2nd for loop goes away as $n^{[0]}=1$, so we get j loop from 1 to n

Forward pass:

$$W^{[1]} = [w_1 \quad w_2 \quad \dots \quad w_n]_{1\times n} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$z = W^{[1]} X + b^{[1]}$$

$$a = g(z)$$

$$\Rightarrow z = \sum_{i=1}^{n} x_i w_i + b.$$

$\frac{\delta L}{\delta w_1}, \frac{\delta L}{\delta w_2}, \dots, \frac{\delta L}{\delta w_n}$ & $\frac{\delta L}{\delta b}$ will be computed by gradient descent.

$$\frac{\delta L}{\delta w_1} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z} \times \frac{\delta z}{\delta w_1}$$

$$\frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z} \times \frac{\delta z}{\delta w_2}$$

So in general,

$$\frac{\delta L}{\delta w_j} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z} \times \frac{\delta z}{\delta w_j} \qquad \forall j \in [1, n]$$

$$\frac{\delta L}{\delta b} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z} \times \frac{\delta z}{\delta b} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z}$$

This is $=1$ as $z = x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b$

Now instead of writing so many eqs, we vectorise it:

$\hat{y} = g(z)$ $\quad z = w^{[1]} x + b$

$z = [\quad\quad]\begin{bmatrix} \\ \\ \end{bmatrix} + b$

$w^{[1]}$ $\qquad\qquad$ x

This is for 1 layer

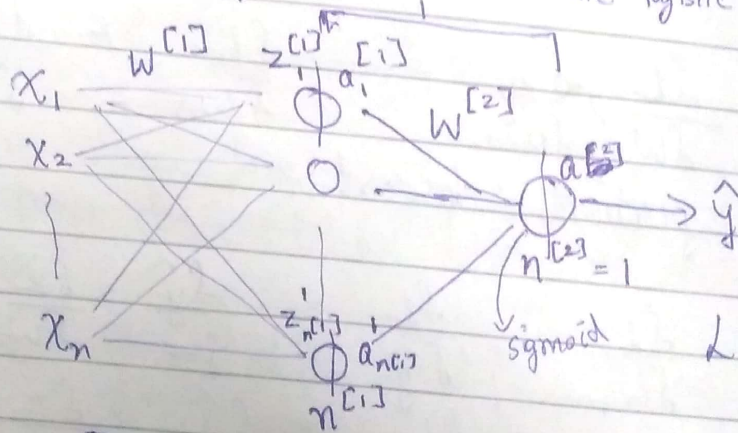
Now we don't to use explicit for loops

$$w^{[1]} = w^{[1]} - \alpha \frac{\delta L}{\delta w^{[1]}}$$

This helps us get rid of 2 for loop. (using example of n I/p)

$$\frac{\delta L}{\delta w^{[1]}} = \begin{bmatrix} \frac{\delta L}{\delta w_1} & \frac{\delta L}{\delta w_2} & \cdots & \frac{\delta L}{\delta w_n} \end{bmatrix}$$

Next generalisation : This part is like logistic regression



$W^{[1]}$ captures all wts at 1st layer

Dimension of $W^{[1]} = n^{[1]} \times n$

" " $W^{[2]} = 1 \times n^{[1]}$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_n^{[1]} \end{bmatrix}_{n^{[1]} \times 1}$$

Dimension of a has to be same as that of z

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_n^{[1]} \end{bmatrix}_{n^{[1]} \times 1}$$

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$

$$\hat{y} = a^{[2]} = g^{[2]}(z^{[2]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$\begin{cases} z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \\ \hat{y} = g^{[l]}(z^{[l]}) \end{cases}$$

We can't get rid of any of the for loops here. (without vectorisation).

Back prop : $\frac{\partial L}{\partial W^{[2]}}, \frac{\partial L}{\partial W^{[1]}}, \frac{\partial L}{\partial b^{[1]}}, \frac{\partial L}{\partial b^{[2]}}$

(so with vectorisation, we don't need any of the for loops)

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial W^{[2]}}$$

If we take the case of logistic regression,

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i} \quad (1)$$

$$L = -(y \log \hat{y} + (1-y) \log (1-\hat{y}))$$

$$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} + \frac{(1-y)}{1-\hat{y}}\right) = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} = \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{(1-\hat{y})\hat{y}}$$
$$= \hat{y} - y / (1-\hat{y})\hat{y} \quad (2)$$

$$\hat{y} = \frac{1}{1+e^{-z}} \Rightarrow \frac{\partial \hat{y}}{\partial z} = \frac{-1}{(1+e^{-z})^2} \times -e^{-z} = \frac{e^{-z}}{(1+e^{-z})^2}$$

This is $g(z)$ - sigmoid

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})}\left(\frac{1+e^{-z}-1}{1+e^{-z}}\right) \quad (2)$$

$$= g(z)(1-g(z))$$

→ This is nothing but $\hat{y}$ as there is only one place where we are using it.

$$\frac{\delta z}{\delta w_i} = \frac{\delta w_i x_i + b}{\delta w_i} = x_i$$

So (1) can be written as:

$$\frac{\delta L}{\delta w_i} = \left(\frac{\delta L}{\delta z}\right) x_i$$

→ This is same irrespective of i.

$\hat{y} = a_1^{[1]}$ for this.

$$\frac{\delta L}{\delta w^{[1]}} = \left[\frac{\delta L}{\delta w_1} \quad \frac{\delta L}{\delta w_2} \quad \cdots \quad \frac{\delta L}{\delta w_n}\right]$$

$$= \left[\frac{\delta L}{\delta z} x_1 \quad \frac{\delta L}{\delta z} x_2 \quad \cdots \quad \frac{\delta L}{\delta z} x_n\right] = \frac{\delta L}{\delta z} \times X^T$$

as X is column matrix

(4)

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z}$$

(from (2) & (3)) $$= \frac{\hat{y}-y}{(1-\hat{y})\hat{y}} \times g(z)(1-g(z))$$

$$= \frac{\hat{y}-y}{(1-\hat{y})\hat{y}} \times \hat{y}(1-\hat{y}) = \hat{y}-y \quad (5)$$

From (4) & (5): $\frac{\delta L}{\delta w^{[1]}} = (\hat{y}-y)X^T$ → Calculate $\frac{\delta L}{\delta b} = \frac{\delta L}{\delta z} = \hat{y}-y$

Now if we take the case of the diagram on prvs page,

L is func. of $W^{[2]}, b^{[2]}, W^{[1]}, b^{[1]}$.

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]}) \longrightarrow \text{Superscript of } g \text{ is a general form if we use more than one activation func.}$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]}) = \hat{y}.$$

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial W^{[2]}}$$

$z^{[2]}$ ↙ as we are doing back prop.

$$\leftarrow \frac{\partial \mathcal{L}}{\partial z^{[2]}} = \hat{y} - y = a^{[2]} - y \quad \text{& similar to prev, } \frac{\partial z^{[2]}}{\partial W^{[2]}} = a^{[1]T}$$

(This is also same as prev derived eq)

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y)\, a^{[1]T}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = a^{[2]} - y.$$

Now if consider the left part of the diagram (diff. from logistic reg.)

$$\leftarrow \frac{\partial \mathcal{L}}{\partial z^{[1]}} = W^{[2]T} \frac{\partial \mathcal{L}}{\partial z^{[2]}} \times g^{[1]}(z^{[1]})$$

This is scalar (1)

↓ Element wise multiplication not matrix

(Dim of this = Dim of $z^{[1]}$ = $n^{[1]} \times 1$)

Final eq.

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} x^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}}$$

Now to get these:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$

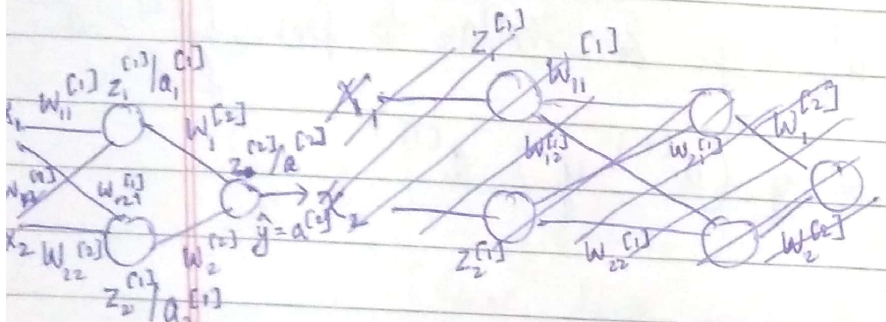
$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}}}_{\frac{\partial \mathcal{L}}{\partial z^{[2]}}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

$$W^{[2]} = \begin{bmatrix} W_1^{[2]} & W_2^{[2]} & \cdots & W_{n^{[1]}}^{[2]} \end{bmatrix}$$

$$W^{[2]T} = \begin{bmatrix} W_1^{[2]} \\ W_2^{[2]} \\ \vdots \\ W_{n^{[1]}}^{[2]} \end{bmatrix}$$

When we multiply this by $\frac{\delta L}{\delta z^{[2]}}$, then we are finding contribution of every neuron i.e propagating the error.

$\Rightarrow$ Thus we get why $W^{[2]T} \frac{\delta L}{\delta z^{[2]}}$



$$z^{[2]} = W^{[2]} a^{[2]} + b^{[2]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$\frac{\delta L}{\delta W_1^{[2]}} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{[2]}} \frac{\delta z^{[2]}}{\delta W_1^{[2]}} \qquad \frac{\delta L}{\delta W_2^{[2]}} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{[2]}} \frac{\delta z^{[2]}}{\delta W_2^{[2]}}$$

$$L = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$
$$\hat{y} = a^{[2]} = g(z^{[2]})$$

① $$\frac{\delta L}{\delta W_{11}^{[1]}} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z^{[2]}} \times \frac{\delta z^{[2]}}{\delta a_1^{[1]}} \times \frac{\delta a_1^{[1]}}{\delta z_1^{[1]}} \times \frac{\delta z_1^{[1]}}{\delta W_{11}^{[1]}}$$

② $$\frac{\delta L}{\delta W_{12}^{[1]}} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z^{[2]}} \times \frac{\delta z^{[2]}}{\delta a_1^{[1]}} \times \frac{\delta a_1^{[1]}}{\delta z_1^{[1]}} \times \frac{\delta z_1^{[1]}}{\delta W_{12}^{[1]}}$$

shorthand for $\frac{\delta L}{\delta z^{[1]}}$ ↙ $$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$ ⑤

③ $$\frac{\delta L}{\delta W_{21}^{[1]}} = \frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z^{[2]}} \times \frac{\delta z^{[2]}}{\delta a_2^{[1]}} \times \frac{\delta a_2^{[1]}}{\delta z_2^{[1]}} \times \frac{\delta z_2^{[1]}}{\delta W_{21}^{[1]}}$$

④ $$\frac{\delta L}{\delta W_{22}^{[2]}} = // \times \cdot \times \frac{\delta z^{[2]}}{\delta a_2^{[1]}} \times \frac{\delta a_2^{[1]}}{\delta z_2^{[1]}} \times \frac{\delta z_2^{[1]}}{\delta W_{22}^{[2]}}$$

Now let us compare ⑤ with ①, ②, ③, ④:
We need only till $\delta z^{[1]}$, so we can ignore last terms of ①, ②, ③, ⑤

First 2 terms of ①, ②, ③, ④,

$$\frac{\delta L}{\delta \hat{y}} \times \frac{\delta \hat{y}}{\delta z^{[2]}} = \frac{\delta L}{\delta z^{[2]}} = dz^{[2]}$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$= \begin{bmatrix} W_1^{[2]} & W_2^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} + b^{[2]}$$

$$\frac{\partial z^{[2]}}{\partial a_1^{[1]}} = W_1^{[2]} \qquad \frac{\partial z^{[2]}}{\partial a_2^{[1]}} = W_2^{[2]} \quad \} \text{ (6)}$$

$$a_1^{[1]} = g_1^{[1]}(z_1^{[1]})$$

$$\frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} = g_1^{[1]'}(z_1^{[1]}) \quad \text{(7)}$$

So from eq (5):

$$\frac{\partial L}{\partial z_1^{[1]}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z^{[2]}}}_{\downarrow} \times \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

$\downarrow$ $\frac{\partial L}{\partial z^{[2]}}$

$\downarrow$ This gives a term in terms of wt (6)

$\searrow$ This gives (7)

- Similar to (5)

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$\Rightarrow \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix}_{2\times 1} = \begin{matrix} W_{11}^{[1]} x_1 + W_{12}^{[1]} x_2 + b_1^{[1]} \\ W_{21}^{[1]} x_1 + W_{22}^{[1]} x_2 + b_2^{[1]} \end{matrix}$$

So even $dz^{[1]}$ must be of dim. 2×1.

Now $dz^{[2]}$ is a scalar $= a^{[2]} - y$ (calculated previously)

$$\frac{\partial L}{\partial z_1^{[1]}} = dz_1^{[1]} = \underbrace{(a^{[2]} - y) W_1^{[2]} \times g^{[1]'}(z_1^{[1]})}$$

→ not matrix mult, element wise mult

$\longrightarrow$

$$\begin{bmatrix} \frac{\partial L}{\partial z_1^{[1]}} \\ \frac{\partial L}{\partial z_2^{[1]}} \end{bmatrix} = \begin{bmatrix} (a^{[2]} - y)\ w_1^{[2]}\ g_1'^{[1]}(z_1^{[1]}) \\ (a^{[2]} - y)\ w_2^{[2]}\ g_2'^{[1]}(z_2^{[1]}) \end{bmatrix}$$

$$= (a^{[2]} - y) \begin{bmatrix} w_1^{[2]}\ g_1'^{[1]}(z_1^{[1]}) \\ w_2^{[2]}\ g_2'^{[1]}(z_2^{[1]}) \end{bmatrix} \quad (8)$$

$$W^{[2]} = \underbrace{\begin{bmatrix} w_1^{[2]} & w_2^{[2]} \end{bmatrix}}_{1\times 2} \Rightarrow W^{[2]T} = \underbrace{\begin{bmatrix} w_1^{[2]} \\ w_2^{[2]} \end{bmatrix}}_{2\times 1}$$

$$(8) = (a^{[2]} - y) \{ W^{[2]T} \overset{*}{\downarrow} \text{element wise mult} \} g'^{[1]}(z^{[1]})$$

$$= (5)$$

If we have L layers, to find $\frac{\partial L}{\partial W^{[1]}}$, we'll have approx. 2 terms per layer i.e we'll have no. of terms of order 2^L.
The problem with this is say if $L = 100$ & we have ~~200~~ around 200 partial der., & say about 100 are ≈ 1 & 100 are ≈ 0.1, then our $\frac{\partial L}{\partial W^{[1]}}$ will become very close to 0

$$\Rightarrow W_{ij}^{(1)} = W_{ij}^{(0)} - \alpha \frac{\partial J}{\partial W_{ij}^{[1]}} \nearrow 0$$

So our G. descent algo will stop updating parameters, so algo will stop as it will think it found optimal values.
There is no solution to this problem (Vanishing Gradient problem)

Similarly if our derivatives > 1, then our $\frac{\partial L}{\partial w_{ij}^{[l]}}$ becomes very large & we will have to subtract a huge number.

exploding gradient problem ← To this problem, we have a solution (gradient clipping), we fix a threshold & everytime our term crosses the threshold, we make the $\frac{\partial L}{\partial w_{ij}^{[l]}}$ = threshold

This happens only with deep NN

Try this with 3 nodes in 1st layer & 1 node in the 2nd layer

For 1 training example:

$$\begin{cases} dz^{[2]} = a^{[2]} - y \\ dW^{[2]} = dz^{[2]} a^{[1]T} \\ db^{[2]} = dz^{[2]} \\ dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) \\ dW^{[1]} = dz^{[1]} x^T \\ db^{[1]} = dz^{[1]} \end{cases} \quad \left(dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]}) \right)$$

$$\left(dW^{[l]} = dz^{[l]} a^{[l-1]} \right)$$

$$\left(db^{[l]} = dz^{[l]} \right)$$

Vectorised form (to get rid of some for loops):

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dz^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

Vectorise last 2 eqs on your own

$$Y = [\underbrace{y^{(1)}}_{1^{st} \text{ training example}} \; y^{(2)} \; \cdots \; y^{(m)}]_{1 \times m}$$

$$A^{[2]}_{(2)} = [a^{[1](1)} \; a^{[1](2)} \; \cdots \; a^{[1](m)}]_{1 \times m}$$

Here we can write $\hat{y}^{(1)} \cdots \hat{y}^{(m)}$ or $a^{[2](1)} \cdots a^{[2](m)}$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \Rightarrow X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{n \times m}$$

n → no. of features

- $\dim(Z^{[2]}) = \dim(dZ^{[2]}) = [\,?\,]_{1 \times m}$

$$Z^{[2]} = \begin{bmatrix} z^{[2](1)} & \cdots & z^{[2](m)} \end{bmatrix}_{1 \times m}$$

$$dZ^{[2]} = \begin{bmatrix} (a^{[2](1)} - y^{(1)}) & \cdots \end{bmatrix}_{1 \times m}$$

$$A^{[1]} = \begin{bmatrix} a^{[1](1)} & \cdots & a^{[1](m)} \end{bmatrix}_{n^{[1]} \times m}$$

when $a^{[1](1)}$ is also written in complete form as it is also a vector

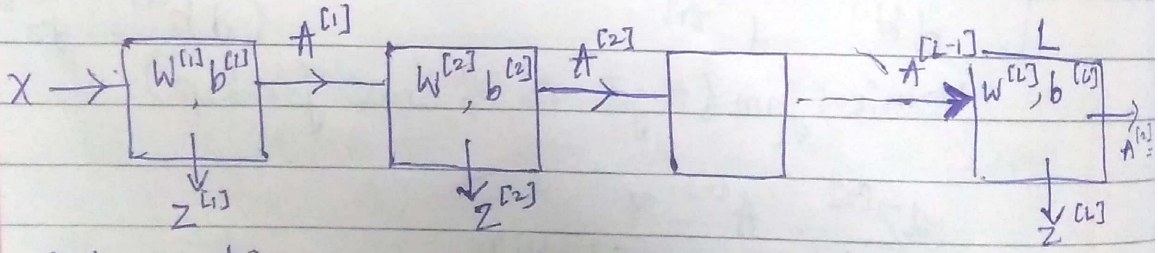
$$A^{[1]T} = \begin{bmatrix} a^{[1](1)} \\ \vdots \\ a^{[1](m)} \end{bmatrix}_{m \times n^{[1]}}$$

$$\dim(W^{[2]}) = \dim(dW^{[2]}) = 1 \times n^{[1]}$$

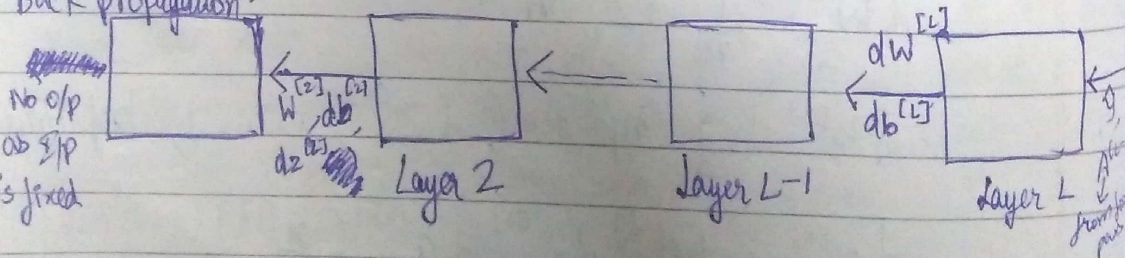
$$dW^{[2]} = \begin{bmatrix} dW_1^{[2]} & dW_2^{[2]} & \cdots & dW_{n^{[1]}}^{[2]} \end{bmatrix}$$

The 1/m factor comes in as we have to sum & avg. over m examples

Forward pass:



Back propagation:

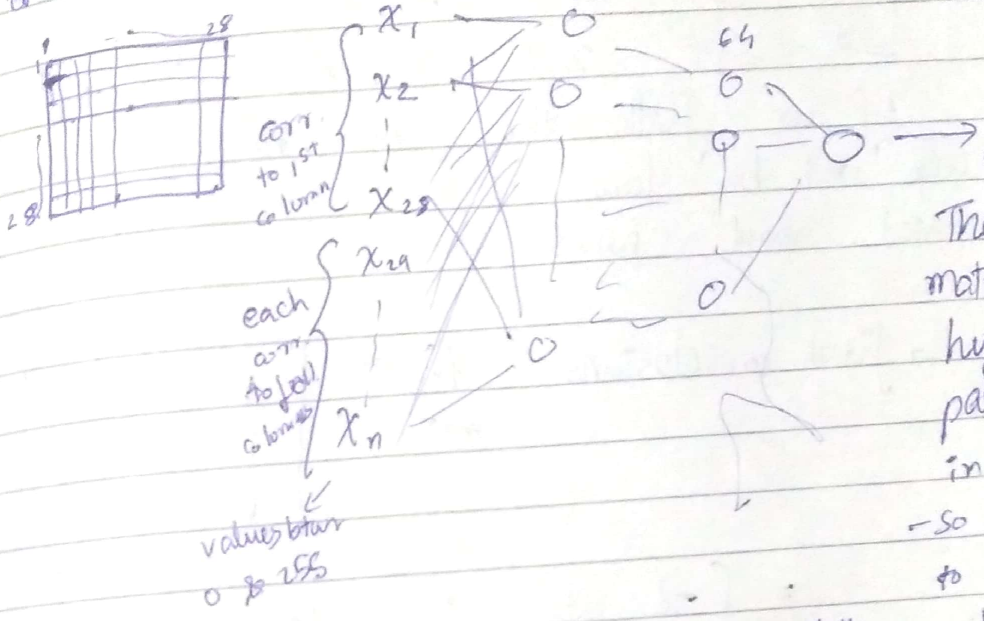


Initialisation of weights: don't give all 0's as then all values of z, a, dz will be zero, don't give all 1's as then all dz values will be same, so we give random values

We want wts to be small, so if we multiply by a small no like 0.001 if range of rand no is large

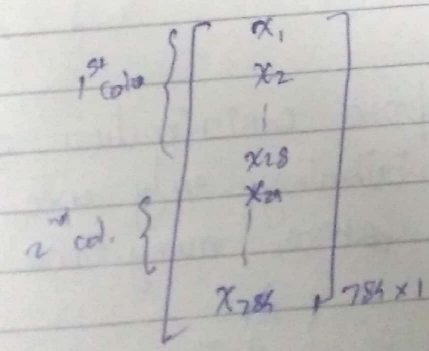
See notes of in btwn class

Say we have 28×28 b/w images which have either 0 or 1. Can we write a binary classifier for this?



The wt. matrix will be huge & many parameters (also in millions)
- so this was thought to be inconvinient.

In tensorflow we have a layer flatten which will put the image in one column matrix.



Using this loses some properties of the image.

CNN's came into use in late 80's when banks needed automatic reading of docs. We can't use above arch as for whom we have huge I/p's, so we use CNN's which give an edge above this.

We want a NN which keeps/preserves local features in an image & uses less no of parameters

Relu puts -ve values as 0

corr. elements $\times 1 + \times 0 + \times -1$

$$A = \begin{bmatrix} \textcircled{5} & 3 & 6 & 8 & 9 & 1 \\ 1 & 6 & 8 & 3 & 2 & 1 \\ 9 & 9 & \textcircled{3} & 2 & 1 & -1 \\ 1 & 7 & 8 & 4 & 6 & 5 \\ 9 & 3 & 4 & 0 & 0 & 1 \\ 0 & 1 & 2 & 3 & 4 & 3 \end{bmatrix}_{6 \times 6}$$

If we flatten this, our no. of parameters will not be low.

Read about CNN's

To find convolusions:

$$\underset{n \times n}{A} * \underset{f \times f}{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}$$

→ filter: helps us differentiate b/wn light & dark place with vertical edge

$$= \begin{bmatrix} -2 & 5 & 5 & 10 \\ -8 & 7 & \cdots & \\ & & & \\ -4 & \cdots & \cdots & -4 \end{bmatrix}_{4 \times 4}$$

→ $(n-f+1) * (n-f+1)$. * See proof of this

4×4 → image size is reducing which is good

Now if we compare contribution of circled 5 & 3, 5 is contributing only once to o/p while 3 is contributing as many times as no. of elements in filter i.e 9.

Now if our image is such if our corner is also imp., above problem will cause an issue, so the easiest way is to pad:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & \text{elements of } A & & & & 0 \\ 0 & & & & & & & 0 \\ 0 & & & & & & & 0 \\ 0 & & & & & & & 0 \\ 0 & & & & & & & 0 \\ 0 & & & & & & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{8 \times 8}$$

$n - f + 1$

$= 8 - 3 + 1$

$= 6$

padding = p

$n' = n + 2p$

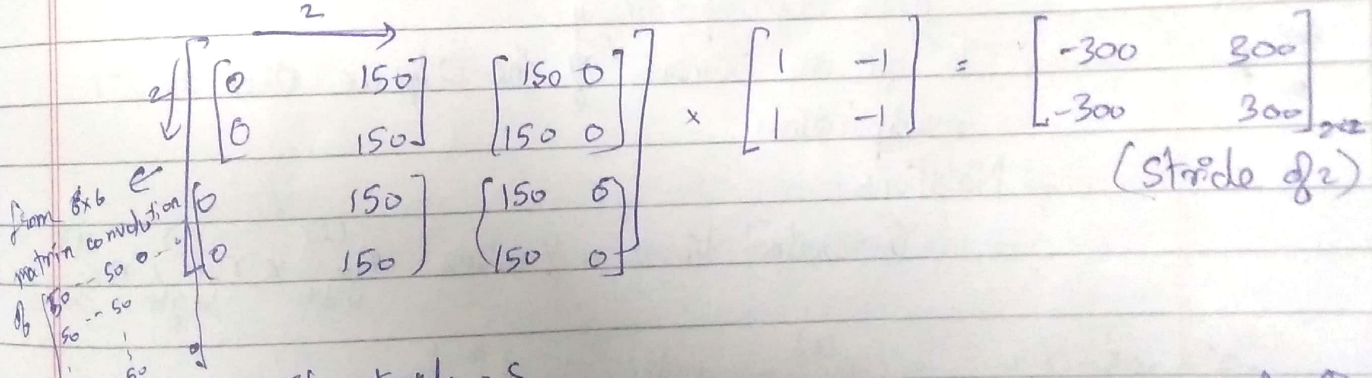
$n_o = n' - f + 1 = n + 2p - f + 1$

Dimension: $(n + 2p - f + 1) \times (n + 2p - f + 1)$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$: verticle edge filter.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$: horizontal edge filter.

Strides:



If stride = s

$$n_o = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$: Proof? ☆☆

Andrew's course gives this formula as:

$$n_o = \left\lfloor \frac{n + 2p - f + 1}{s} \right\rfloor$$

But if we take $p = 0$, ~~$s = 2$~~ $s = 4$ $(s > 2)$,

$n = 44$, $f = 3$

① $n_o = \left\lfloor \frac{44 - 3}{4} \right\rfloor + 1 = 11$ → This was found to be better

② $n_o = \left\lfloor \frac{44 - 3 + 1}{4} \right\rfloor = 10$ (no fundamental reason)

, channel: gray coloured

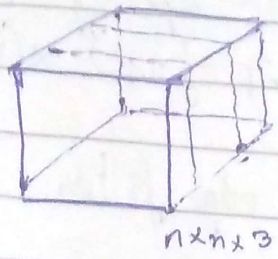
Say we have a $n \times n \times 1$ image

Coloured image: (R, G, B) : 24 bits

0 - 255

3 channels

i.e 3 2-D matrices put one over other



How to convolve?

Filter: $3 \times 3 \times 1$ $\Rightarrow$ O/p dimension: $4 \times 4 \times 3$

There is nothing fundamentally wrong with this but we don't use this as below one performs better.

If our filter was $3 \times 3 \times 2$, then o/p dim will be $4 \times 4 \times 2$.

If our filter: $3 \times 3 \times 3$ $\Rightarrow$ O/p dimension: $4 \times 4 \times 1$

To convolve we multiply $R_1^{G,B}$ channel of I/p & filter correspondingly & get one element of the O/p we do 27 multiplications.

Notations:

Generalised dimension of volume: $n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$ (width, height, channel)

Even filter size used in padding.

$p^{[l]}$ - padding in l^{th} layer

$s^{[l]}$ - stride " "

$f^{[l]}$ - dimension of filter " "

So in:

$$\underbrace{6 \times 6 \times 3}_{n_w^{[0]}\ n_H^{[0]}\ n_c^{[0]}} * \underbrace{3 \times 3 \times 3}_{f^{[1]}\ n_c^{[l-1]} \text{ or } n_c^{[0]}} = \underbrace{4 \times 4 \times 1}_{n}$$

$p^{[0]}$ doesn't make sense, padding done in I/p is $p^{[1]}$

$p^{[1]} = 0$, $s^{[1]} = 1$

If we use 4 diff. filters of same dimensions:

$6 \times 6 \times 3$ * f_1 $(3 \times 3 \times 3)$ = $4 \times 4 \times 1$
* f_2 $4 \times 4 \times 1$
* f_3 $4 \times 4 \times 1$
* f_4 $4 \times 4 \times 1$

Every ~~filter~~ filter will highlight a diff. aspect of the image. (horizontal lines, vertical lines etc), but it will be hard to keep track so we overlap them all to get a $4 \times 4 \times 4$ o/p.

$n_w^{[l-1]} \times n_H^{[l-1]} \times n_c^{[l-1]}$ — $f^{[l]}$, $p^{[l]}$, $s^{[l]}$ → $n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$

o/p of prvs layer

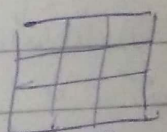
Actual dim. of filters: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

$n_c^{[l]}$ = no. of filters in l^{th} layer

From above example: $4 \times 4 \times 4$

Convolution is a linear operation (it is equivalent to $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$ - we are yet to add bias)

$n_w^{[l-1]} \times n_H^{[l-1]} \times n_c^{[l-1]} \longrightarrow n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$



To add bias we add $b^{[l]}$ to all elements in the o/p matrix.

$$b^{[l]} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}_{\#\text{filter} \times 1}$$

So we have completed the Z part i.e linear part. Our parameters are incorporated in the filters. They are learnt.

$$\begin{bmatrix} W_1 & W_2 & \cdots \\ & & \\ & & \end{bmatrix}$$

If we don't have non-linearity in every layer, then we can just remove layers & compress to one as linear func of linear func is a linear fn.

To find $A^{[l]} = g^{[l]}(z^{[l]})$

ReLU $\left(n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}\right) = n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$

↓

Except case of ReLU, all values $< 0 = 0$, otherwise same. In case of image value $> 255 = 255$

In practice we normalise the values so > 255 won't arise.

CNN's are closer to the vision part of our networks. Our conscious mind can process 50 b/s, our senses can percieve 11 MB/s.

Pooling: Suppose $p=0$, $s=2$, filter $= 2$.

In case of max pooling:

$$\begin{bmatrix} \begin{bmatrix} 8 & 4 \\ 9 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 2 & 8 \end{bmatrix} \\ \begin{bmatrix} 1 & 5 \\ 2 & 0 \end{bmatrix} & \begin{bmatrix} 7 & 1 \\ 3 & 7 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 9 & 8 \\ 5 & 7 \end{bmatrix}_{2\times 2}$$

We can take any other func other than max., chosen empirically.

Pooling is done only after convolution by convention.
Translation is invariant in pooling (to some extent).
Rotation is not " " " but you can make it learn that invariance. There are no learning parameters in pooling.

Say we have a $100 \times 100 \times 3$ & we use 10 filters of $3 \times 3 \times 3$ to do 1 convolution + pooling. What is the no. of learnable parameters?

I/p has no parameters.

Filter has $(27 \times 10) + 10$ (bias) parameters

$= 280$

Computational cost & O/p size will size ↑

If we change image size to 200 × 200 × 3 i.e 4 times the size, using CNN the no of parameters will still remain same i.e it is invariant to pic size while tho if we were using our normal DNN, we would have a massive increase in no of parameters. That's why in images we use CNN as I/p size is huge.

Suppose we have a 6×6 b/w image which we flatten & say we have 5 neurons in the 1st layer i.e $W^{[1]} = 5 \times 36$