DL

Epoch :- number of times data traversed for optimizing parameters.

Hidden layers minimizes the error function & increases optimization & extract features from the linear classifier inputs.

Q What does a single neuron do?

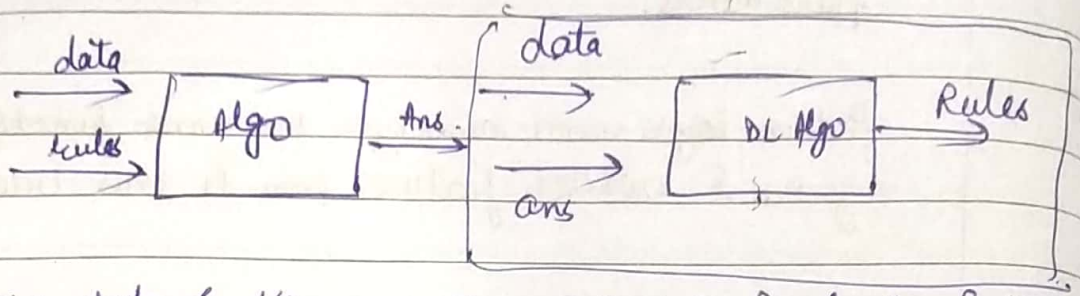
* adding non trivial non linearity.

If we want linear separation, there is no need of neurons or separate layers.

Test loss → (misclassification)
Training loss → less than loss (misclassification).
Training set →

Image Net Dataset :-

Tensorflow Specialization

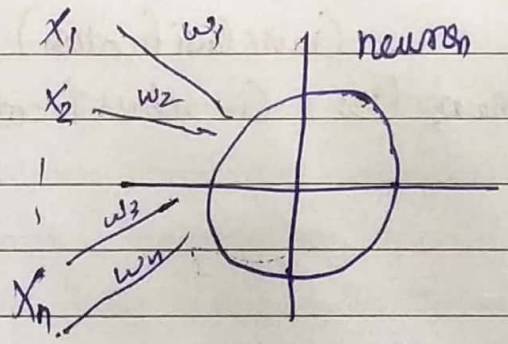


given the data & the answer, we try to find the pattern.

* representation of data in such a manner that it learns ~~learn~~ & it learns function efficiently. (more the data better prediction of function)
* We talk mostly of supervised learning.

What does a neuron do?

$y = wx$.

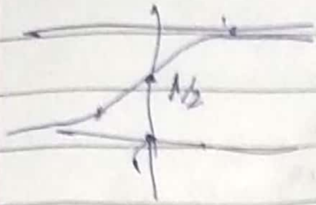


$$y = \sum_{i=1}^{n} x_i w_i.$$

$$y = f \circ (g \circ (h\,x))$$

$$y = \sigma\left(\sum_{i=1} x_i \cdot w_i\right)$$

sigmoid f^n. with domain $(-\infty, \infty)$ with range $(0, 1)$



Keywords

model :-

Sequential :- put all layers sequentially (i.e. all layers one ~~of~~ after other).

```
import tensorflow as tf
import numpy as np
from tensorflow import keras.

model = ~~tf.~~ tf.keras.Sequential ([keras.layers.Dense
                                      (units =1, input_shape=[1])
```

what kind of input shape receiving.

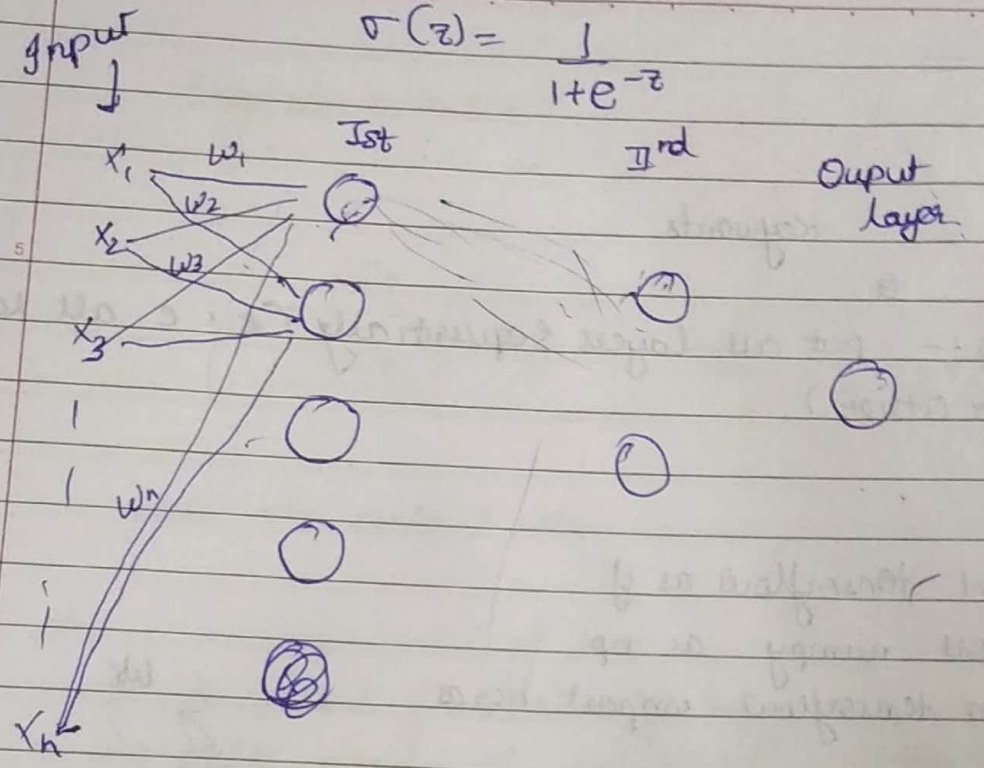
it runs till it reaches a minimum ∈.

```
model.compile (optimizer = 'sgd', loss = 'mean_squared_error')

model.fit (x, y, epochs=10)
print (model.predict ([]))
```

stochastic Gradient descent is used to minimize the cost function, which is calculated on the basis of input features. Main aim is to extract important features.

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



If there are 4 neurons & n data points then there are 4n parameters.

$$w^{[1]} = \begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$$

this represents [0-1] matrix n x m where m is number of neurons layer.

now matrix size for a general l^{th} layer would be.

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

$n^{[l]}$ → number of neurons in l^{th} layer

$n^{[l-1]}$ → no. of neurons in $(l-1)^{th}$ layer

Every layer will have input, (weights) $W^{[i]}$ & number of neurons $n^{[1]}$. & two outputs, one denoted by z & the other one by a

$z_3^{[1]}$, $a_3^{[1]}$

→ first → third

In third layer first neuron. In first layer third neuron.

For a single layer, weight matrix is represented as

To create a weight matrix for each neuron in a layer, we will create a vector.

Output of each layer will be defined as.

(linear output i.e. z)

for a single neuron

$$\begin{bmatrix} w_{11}^{[1]} & \cdots & w_{1n}^{[1]} \\ w_{21}^{[1]} & \cdots & w_{2n}^{[1]} \\ w_{31}^{[1]} & \cdots & w_{3n}^{[1]} \\ w_{41}^{[1]} & \cdots & w_{4n}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

↙ Output from first layer.

$$A^{[1]} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \sigma(z_4^{[1]}) \end{bmatrix} \rightarrow \begin{matrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{matrix}$$

$z^{[1]}$ Now output of first layer is input of second layer

Now z, A & W for ~~second to first to~~ second layer

$$W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & \cdots & w_{14}^{[2]} \\ w_{21}^{[2]} & \cdots & w_{24}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

(Linear output)

$$z^{[2]} = W^{[2]} \cdot A^{[1]}$$

$$A^{[2]} = \sigma(z^{[2]})$$

$$z^{[l]} = W^{[l]} A^{[l-1]}$$

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

Conv (Input Data & rules)

DL (Input Data & answers) expect ~~Ans~~ rules

Camlin | Page | Date / /

For third layer. $W^{[3]} = [\,]_{1 \times 2}$

$z^{[3]}$

$A^{[2]}$ → Finally is the

$f(x,y,z) = x^2+y^2+z^2-t^2 = 0$

Neuron that computes only linear thing → the one shared in Colab.

Other than that all neurons compute linear & non linear.

* Units :- no. of neuron in the layer.
* input shape = [,]

Single neuron with multiple input performs similarly to logistic regression

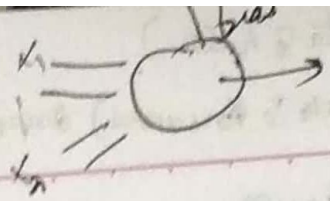
For eg: An image of a cat with 40K pixels.

Each pixel will be a separate input to a neuron

Now suppose we have 100K images each of 40K pixels.

actual data	I_1	Y
	I_2	N
	⋮	Y
	⋮	
	I_{10K}	Y

There are 40K parameters (each pixel is param)



$$z^{[1]} = w^{[1]} a^{[0]} + b.$$

$$\hat{y} = \sigma(z^{[1]})$$

assigning huge loss to wrong predictions i.e. upto ∞.

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

actual value (0 or 1) — prob.

Consider $y=1, \hat{y}=1, L=0$

$y=0, \hat{y}=0, L=0$

y = actual ~~for prediction~~ value

$\hat{y}$ = predicted value

Cost function = addition of all losses, then avg.

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L^{(i)}(y^{(i)}, \hat{y}^{(i)})$$

where m is number of losses

if all losses are 0, cost will also be 0.

Dimensions (parameters) in this case are 40 k+1, (40 k are pixels & 1 is extra

Gradient η

$$w_1 = w_1 - \alpha \frac{\partial J(w,b)}{\partial w_1}$$

α is learning rate.

$$w_2 = w_2 - \alpha \frac{\partial J(w,b)}{\partial w_2}$$

positive slope so ∴ downward.

$$w_{40,000} = w_{40,000} - \alpha \frac{\partial J(w,b)}{\partial w_{40,000}}$$

$$b = b - \alpha \frac{\partial J(w,b)}{\partial b}$$

initially

We take ~~#~~ random values of $w_1, w_2 \ldots w_n$ & then for each parameter update it such that at last we have minimum value for each parameter.

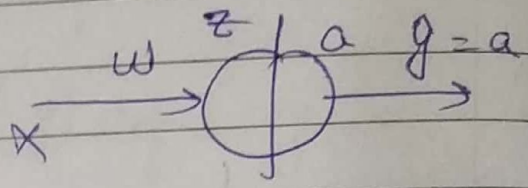
Vanilla version of gradient descent

* bias is some weightage which is not associated with the input.

Date / /

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

$$w = w - \alpha \frac{dJ}{dw}$$



$$\frac{dL}{dw} = \frac{dL}{d\hat{y}}$$

$$\frac{dL}{dw} = \frac{dL}{d\hat{y}} \times \frac{d\hat{y}}{dz} \cdot \frac{dz}{dw}$$

$$\frac{dL}{d\hat{y}} = -\left(\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}(-1)\right)$$

$$= \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

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

$$\frac{d\hat{y}}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = \hat{y} - \hat{y}^2$$

$\frac{dz}{dw} \neq$ Since x is a single input so z is $= xw$.

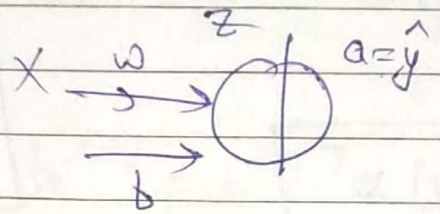
$$\frac{dz}{dw} = x$$

$$\frac{dL}{dz} = \hat{y} - y. = (a-y)$$

$$\frac{dL}{dw} = \frac{dL}{dz} * \frac{dz}{dw}$$

$$= (a-y) * x$$

Back propagation → Chain rule



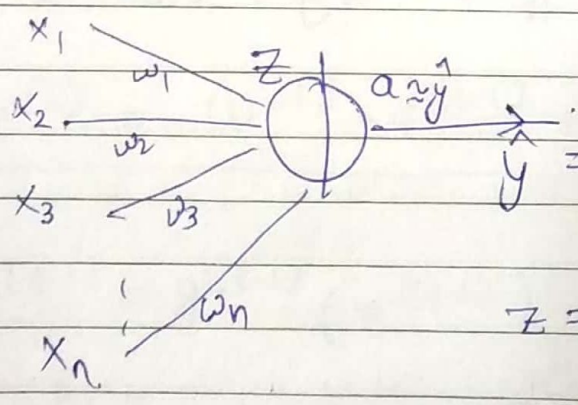
Now for b.

$$\frac{dL}{db} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dz} \frac{dz}{db}$$

$z = w \cdot x + b$

$$\frac{dL}{db} = (\hat{y} - y)$$

* Back propagation logistic regression



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

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

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

$$\frac{dL}{dw_i} = w_i - \alpha \frac{dL}{dw_i}$$

$$z = x_1 w_1 + \_\_\_\_ + x_n w_n + b.$$

$$\frac{dz}{dw_1} = x_i \frac{dL}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw_i}$$

$$\frac{dL}{dw} = (\hat{y} - y) \frac{dz}{dw} \qquad w = [w_1, w_2 \dots w_n]$$

$(\hat{y} - y)$ ↓ scalar

$$\frac{dL}{dw} = (\hat{y} - y)\, x \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\boxed{\frac{dL}{dw} = (\hat{y} - y)\, x^T}$$

Forward & Backward

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

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

Gradient Descent :-

Repeat → till loss change stops

for $l = 1$ to L

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

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

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

$$b_i^{[l]} = b_i^{[l]} - \alpha \frac{\partial J}{\partial b_i^{[l]}}$$

→ Obtained through back propagation.

Q Why we use gradient descent?

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

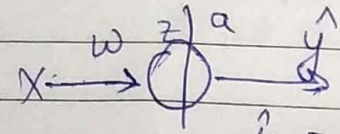
$$\hat{y} = a^{[L]} = g^{[L]}(z^{[L]})$$

$$z^{[L]} = w^{[L]} \cdot a^{[L-1]} + b^{[L]}$$

→ activation function at layer L.

$$a^{[l-1]} = g^{[l-1]}(z^{[l-1]})$$

$$z^{[l-1]} = w^{[l-1]} A^{[l-2]} + b^{[l-1]}$$



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

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

$\frac{dL}{d\hat{y}} \times \frac{d\hat{y}}{dz} \times \frac{dz}{dw}$

y_i

Camlin Page
Date / /

repeat → update converges.

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

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

~~$(y, \hat{y})$~~

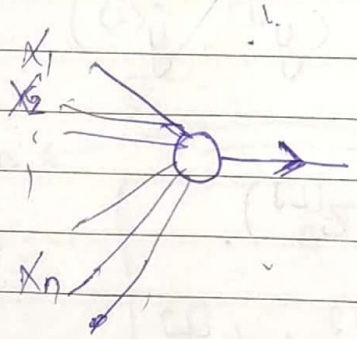
you don't have control over $\hat{y}$.

1 training ex:-

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

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

$$\frac{dL}{dw} = \frac{dL}{d\hat{y}} * \frac{d\hat{y}}{dz} * \frac{dz}{dw}$$



Now for more than 1 training example.

w = no of neurons in (dimension) next layer × no of neurons in previous layer

Put $\hat{y}$ instead of t

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

Forward pass

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

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

Then when we get $\hat{y}$, & loss is high we go for backward pass.

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots \frac{\partial L}{\partial w_n}, \frac{\partial L}{\partial b}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z} * \frac{\partial z}{\partial w_1},$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z} * \frac{\partial z}{\partial w_2}.$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z} * \frac{\partial z}{\partial w_i}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z} * \frac{\partial z}{\partial b} \rightarrow 1.$$

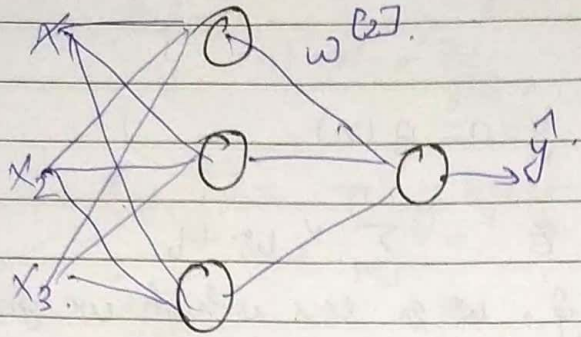
(3) Vectorization.

$$\frac{\partial L}{\partial w^{[1]}} = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2} \dots \frac{\partial L}{\partial w_n} \right].$$

Instead of $w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$

write $\underset{1 \times n}{W^{[1]}} = \underset{(1 \times n)}{W^{[1]}} - \alpha \left[\frac{\partial L}{\partial W^{[1]}} \right]_{1 \times n}$

$w^{[1]}$ $\frac{\partial - y}{\hat{y}(1-\hat{y})}$ $1-\hat{y}$ $\hat{y}(1-\hat{y})$



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

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ \vdots \\ z_n^{[1]} \end{bmatrix}$$

$\hat{y} = a^{[2]}$

(Forward Backward pass for training example)

$$a^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \\ \vdots \\ a_n^{[2]} \end{bmatrix}$$

Camlin Page
Date / /

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

$$a^{[1]} = g^{[1]}(z^{[1]})$$

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

Recap

$z = \sum w_i x_i + b$

$a = \sigma(z)$

Why do we need backward pass:-
to calculate $\frac{\partial J}{\partial w_{ij}}$ or $\frac{\partial L}{\partial w_{ij}}$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$w^{[1]} \frac{\partial L}{\partial w^{[1]}}$

$w^{[1]} = [w_1, w_2, \dots w_n]$. (representation in matrix form)

Purpose of back propagation

Purpose is to tell network layers by layers from last to first that they are wrong.
finally change your parameters accordingly.

$$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} \cdot \frac{(1-y)}{(1-\hat{y})}\right)$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{+1}{(1+e^{-z})^2} \times e^{-z}$$

$$\frac{1+e^{-z}-1}{1+e^{-z}} \cdot \frac{1}{(1+e^{-z})}$$

$$\frac{\partial \hat{y}}{\partial z} = \left(\frac{1}{1+e^{-z}}\right) * \left(\frac{(1+e^{-z}-1)}{(1+e^{-z})}\right)$$

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

$\frac{\partial z}{\partial w_i}$

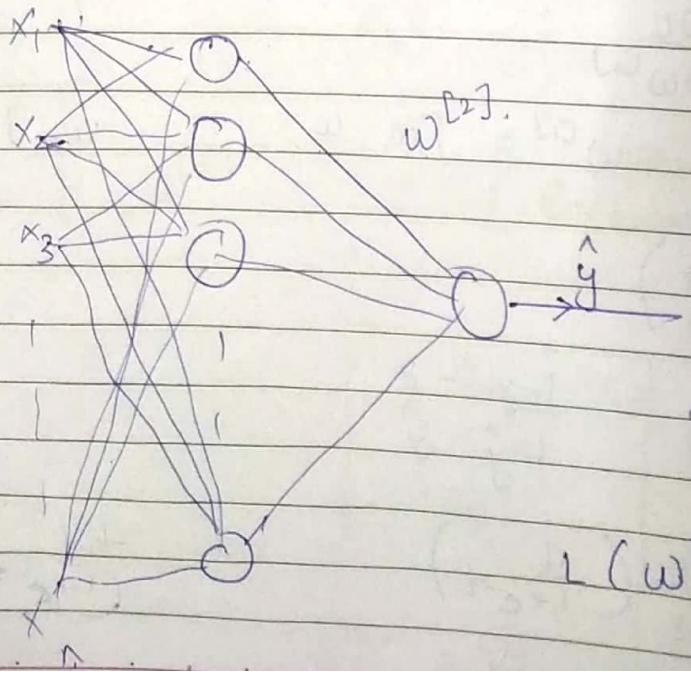
$$\frac{\partial z}{\partial w_i} = x_i.$$

$\Rightarrow$ $\frac{\partial L}{\partial w_i} = \left(\frac{\partial L}{\partial z}\right) x_i$ → we dubbed it because.

Trying to use matrices (vectorizing).

$$\frac{\partial L}{\partial w^{[1]}} = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots \frac{\partial L}{\partial w_n}\right]$$

$$\left[\frac{\partial L}{\partial z} x_1, \frac{\partial L}{\partial z} x_2 \ldots \frac{\partial L}{\partial z} x_n\right]$$



Eq 1.

$L(w^{[2]}, b^{[2]}, w^{[1]}, b^{[1]})$.

$$\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{(1-\hat{y})\,\hat{y}}$$

Now we want $\left(\frac{\partial L}{\partial \hat{y}}\right)\left(\frac{\partial \hat{y}}{\partial z}\right)$

$$= \frac{\hat{y} - y}{(1-\hat{y})\,\hat{y}} * \left(\hat{y}(1-\hat{y})\right) = \hat{y} - y$$

$$\frac{\partial L}{\partial w^{[1]}} = (\hat{y} - y)\, X^T$$

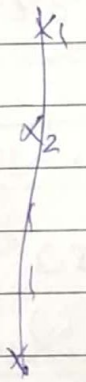
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

~~logg~~ Logistic regression.

dimension of w is 1×n

so we have to take X^T to multiply with $\hat{y} - y$.

$$\frac{\partial L}{\partial b} = \hat{y} - y$$



Forward propagation for fig 1

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

$$a^{[1]} = g(z^{[1]})$$

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

$$a^{[2]} = g^{[2]}(z^{[2]})$$

we can use different activation functions at diff layers, diff

Backward propagation

$$\frac{\partial L}{\partial w^{[2]}} = \left(\frac{\partial L}{\partial z^{[2]}}\right) \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}}$$

$$= (\hat{y} - y)$$

$$= (a^{[2]} - y)\ a^{[1]T}$$

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

$$w_2^{[2]} = w_2^{[2]} - \alpha \frac{\partial L}{\partial w_2^{[2]}}$$

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

Now moving a step back

$$\frac{\partial L}{\partial w^{[1]}} = \left[\frac{\partial L}{\partial z^{[1]}}\right] * \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

$$\frac{\partial L}{\partial z^{[1]}} = w^{[2]T}\left(\frac{\partial L}{\partial z^{[2]}}\right) * g^{[1]}(z^{[1]})$$

Looking for its derivation

Derivation

$$\frac{\partial L}{\partial z^{[1]}} = \left(\frac{\partial L}{\partial a^{[1]}}\right)\left(\frac{\partial a^{[1]}}{\partial z^{[1]}}\right)$$

$$\frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{12}}$$

Summary of gradient descent.

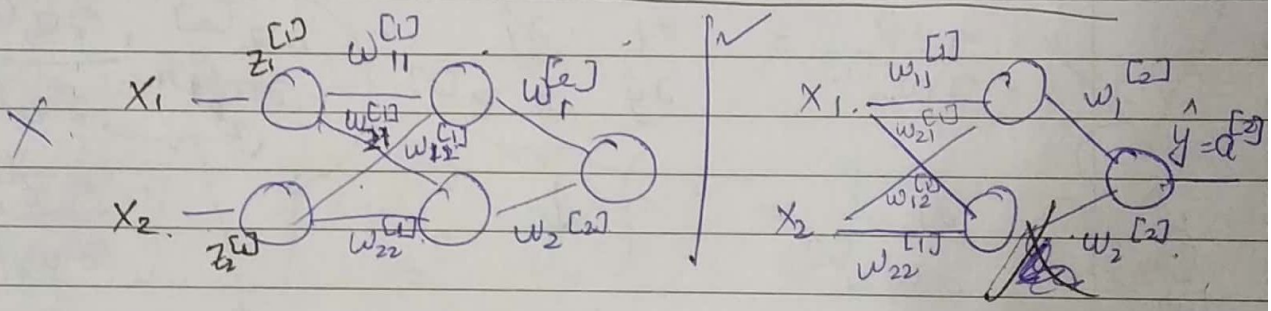
Camlin Page
Date / /

Now how to derive this

$$\cancel{w^{[2]T}} \; z$$

$$w^{[2]T} = \frac{\partial L}{\partial z^{[2]}} \begin{bmatrix} w_1^{[2]} \\ w_2^{[2]} \\ \vdots \\ w_n^{[2]} \end{bmatrix} * \begin{bmatrix} g_1^{[1]}(z_1^{[1]}) \\ \vdots \\ g \end{bmatrix}$$

$$w^{[2]} = [w_1^{[2]} \; w_2^{[2]} \; -- \; w_n^{[2]}]$$



$$\frac{\partial L}{\partial w^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}} \quad \cdots (*)$$

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

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

$$\frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

This is final ans :-

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$$dz^{[1]} \rightarrow \frac{\partial L}{\partial z^{[1]}}$$

Camlin Page
Date / /

$$\frac{\partial L}{\partial w_1^{[2]}} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w_1^{[2]}}$$

$$\frac{\partial L}{\partial w_2^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_2^{[1]}} \cdot \frac{\partial a_2^{[1]}}{\partial z_2^{[1]}} \cdot \frac{\partial z_2^{[1]}}{\partial w_2^{[2]}}$$

$$\frac{\partial L}{\partial w_2^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w_2^{[2]}}$$

$$\frac{\partial L}{\partial w_{11}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{11}^{[1]}}$$

$$\frac{\partial L}{\partial w_{12}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_2^{[1]}} \cdot \frac{\partial a_2^{[1]}}{\partial z_2^{[1]}} \cdot \frac{\partial z_2^{[1]}}{\partial w_{12}^{[1]}}$$

$$\frac{\partial L}{\partial w_{21}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{21}^{[1]}}$$

$$\frac{\partial L}{\partial w_{22}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_2^{[1]}} \cdot \frac{\partial a_2^{[1]}}{\partial z_2^{[1]}} \cdot \frac{\partial z_2^{[1]}}{\partial w_{22}^{[1]}}$$

we are finding $\frac{\partial L}{\partial z_1^{[1]}}$

Now we want final ans

$$dz^{[1]} = w^{[2]T} \cdot dz^{[2]} * g^{[1]'}(z^{[1]})$$

element wise

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

$$= \begin{bmatrix} w_1^{[2]} & w_2^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} + b^{[2]}$$

$w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]}$

$$\frac{\partial z^{[2]}}{\partial a_1^{[1]}} = w_1^{[2]}$$

$$\frac{\partial z^{[2]}}{\partial a_2^{[1]}} = w_2^{[2]}$$

← This is the idea where the terms can come from.

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$\frac{\partial z^{[2]}}{\partial a_1^{[1]}}$

$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}}$

Now we will look exactly where these terms comes from

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

we are finding $\frac{\partial L}{\partial z_1^{[1]}}$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}}$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{matrix} w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + b_1^{[1]} \\ w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + b^{[1]} \end{matrix}$$

element wise

$$\frac{\partial L}{\partial z_1^{[1]}} = dz_1^{[1]} = (a^{[2]} - y) \cdot w_1^{[2]} \cdot g'(z_1^{[1]})$$

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

$\sigma(z) \quad \frac{1}{1+e^{-z}}$ 9

Camlin Page
Date / /

Try this again.

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

$$= (a^{[2]} - y) \begin{bmatrix} w_1^{[2]} \\ w_2^{[2]} \end{bmatrix} * \begin{bmatrix} g'(z) \\ g'(z) \end{bmatrix}.$$

$$w^{[2]} = \begin{bmatrix} w_1^{[2]} & w_2^{[2]} \end{bmatrix}$$

$$w^{[2]^T} = \begin{bmatrix} w_1^{[2]} \\ w_2^{[2]} \end{bmatrix}$$

Back propogation → If any of the derivative is small.

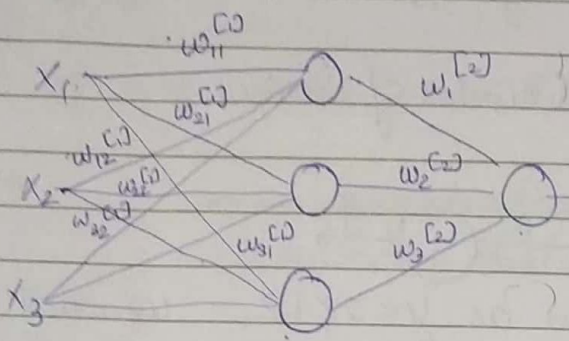
$$w_{ij}^{[2]} = w_{ij}^{[2]} - \alpha \frac{\partial J}{\partial w_{ij}^{[2]}}$$

→ Vanishing gradient problem ⟶ ✗ no soln
→ exploding gradient problem
↓
Solution to this is gradient clipping

If gradient comes close to 0, it might not be the optimal soln but your learning will stop.

H.W. 3 nodes in first layer, 1 node in last layer with 3 inputs

$z = wx + b$ $\quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}$



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

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

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

1×3

$W^{[2]} = \begin{bmatrix} w_1^{[2]} & w_2^{[2]} & w_3^{[2]} \end{bmatrix}$

$W^{[2]T} = \begin{bmatrix} w_1^{[2]} \\ w_2^{[2]} \\ w_3^{[2]} \end{bmatrix}$ $\begin{bmatrix} g'(z_1^{[1]}) \\ g'(z_2^{[1]}) \\ g'(z_3^{[1]}) \end{bmatrix}$

$\frac{\partial L}{\partial w_{11}^{[1]}} = \left[\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \right] \cdot \frac{\partial z_1^{[1]}}{\partial w_{11}^{[1]}}$

$\frac{\partial L}{\partial w_{21}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_2^{[1]}} \cdot \frac{\partial a_2^{[1]}}{\partial z_2^{[1]}} \cdot \frac{\partial z_2^{[1]}}{\partial w_{21}^{[1]}}$

$\frac{\partial L}{\partial w_{12}^{[1]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{12}^{[1]}}$

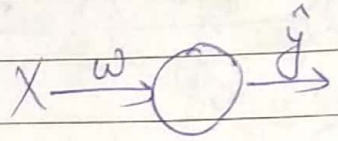
$(a^{[2]} - y) * g^{[1]'}(z_1^{[1]})$

$z^{[2]} = W^{[2]} \cdot X + b^{[2]}$

#Code

nof(.keras.layers.Dense) are no of layers.

units = 1 (number of neurons).



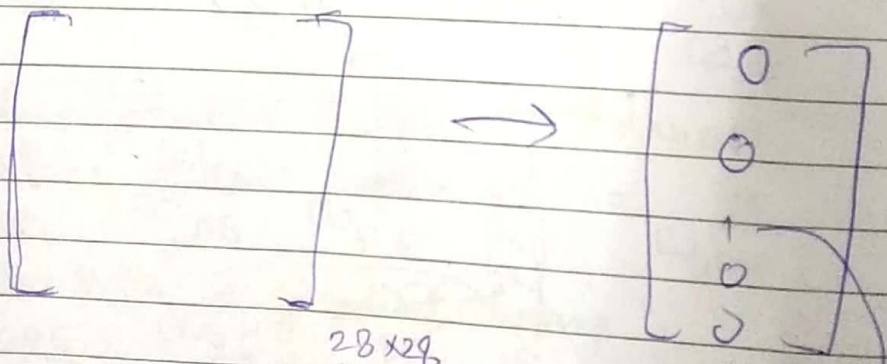
f^n can be written as $y = 2x + 2$. We want our neural network to learn the function.

model.compile (optimizer = 'sgd', loss = 'mean_squared_error')

model.fit (~~fit~~ giving input & output trying to learn the fn.

#Fashion MNist data

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



Dense suggest that every neuron in the layer have inputs from all features.

* keras.layers.Dense means all inputs are connected to this layer (each neuron)

* 90% of time we use relu (activation function) → hidden layer

loss fn ← binary classification → softmax & sigmoid
multiclass classification → softmax

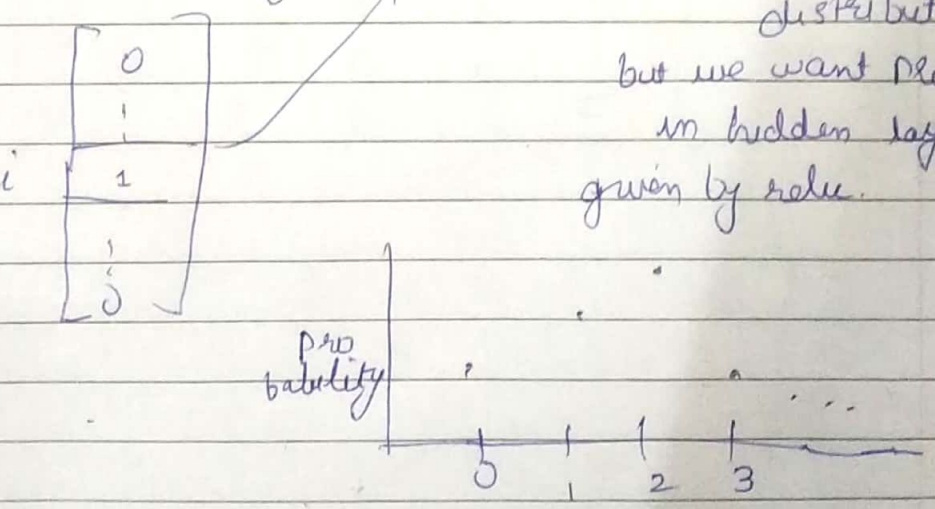
* tf.keras.models.Sequential ( )
↓
all these layers are executed sequentially

Q Why can't we use softmax as activation fn in hidden layer

We can use but problem is.

Softmax looks like $\frac{e^{x_i}}{\sum e^{x_i}}$ $\frac{e^i}{\sum_i e^i}$ → will give probability distribution.

but we want probability in hidden layer given by relu.



1 epoch :- do it once for whole data set, it is 1 epoch.

Initially there was a while loop that runs until it updates all. To achieve this, we use epochs which ensures that no. we will run till epochs (whatever the no be)

* Solve exercises from jupyter

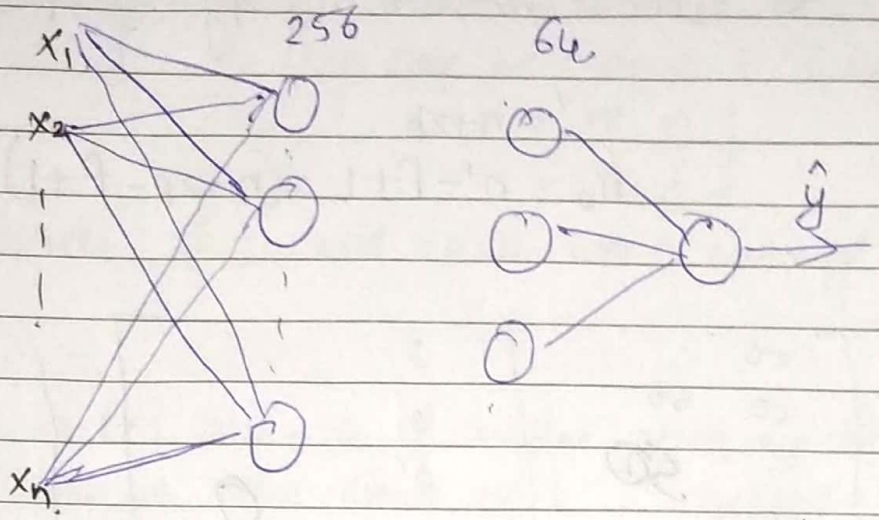
* Model starts learning from where it was left.

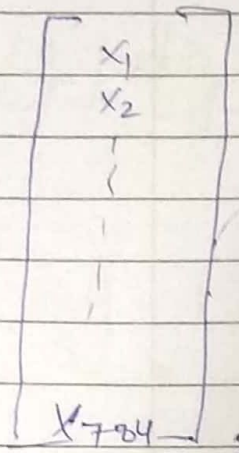
what is use of CNN?

Nothing special but will be able to do same thing with less parameters.

One thing CNN do is → don't flatten the image like that of deep neural networks.

tf.keras.dense.flatten → what it does is take the 28x28 image & flatten it.





→ ~~Keeps~~ / preserve local features in an image.
→ Less # parameters.

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

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

nxn matrix with a fxf filter → $(n-f+1)\ (n-f+1)$.

If we want to increase contribution from 5, we pad 0's everywhere.

Padding P will result into new dimension $n' = n+2p$

Vertical filter lets you recognize vertical edges

~~Solve exercises on jupyter~~ if any.

$n' = n + 2p$

$n_0 = n' - f + 1 = (n + 2p - f + 1) \times (n + 2p - f + 1)$

$$\begin{bmatrix} 50 & & & 0 \\ 50 & 50 & & 0 \\ 50 & 50 & & 0 \\ 50 & & & 0 \\ & & & 0 \\ & & & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

If transition from light to dark $\rightarrow$

$$\begin{bmatrix} 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & & & 0 \\ 0 & & & 0 \end{bmatrix}$$

If transition from dark to light

$$\begin{bmatrix} 0 & -150 & -150 & 0 \\ 0 & & & 0 \\ 0 & & & 0 \\ 0 & & & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \end{bmatrix} \quad 4 \times 4 \qquad \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} -300 & 0 & 300 \\ -300 & 0 & 300 \\ -300 & 0 & 300 \end{bmatrix} \quad 3 \times 3$$

But suppose if we want to stride

$\left[\frac{n-f+1}{s}\right] \quad \left[\frac{n-f}{s}\right]+1$

$\left(\frac{n-f}{s}+\frac{1}{2}\right)$

In that case we want 2 × 2 matrix

$\left[\frac{n}{f}\right]$

Stride of 2 will result into a matrix of $\left[\frac{n}{f}\right] \times \left[\frac{n}{f}\right]$

n-f+1 was size of matrix after a filter.
now we want stride of s. $\left[\frac{n-f}{s}\right]+1$

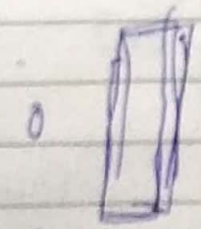
$\left[\frac{5-2}{2}\right]+1$

So if we want stride

$$\begin{bmatrix} 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & 150 & 150 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}.$$

$$= \begin{bmatrix} \quad \end{bmatrix}_{2\times 2}$$

Stride is should be less than filter.



bar.
(cannot be distinguished)

So if size is less than overlapping will be there

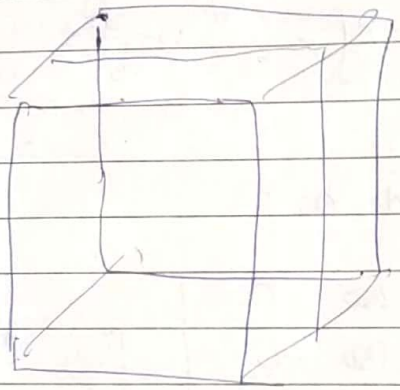
Convolution in colored image

(R, G, B) → 3 bytes

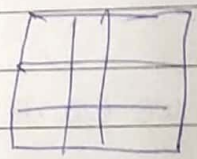
n × n × 1 → black & white image

n × n × 3 → colored image

Q How to take convolution for colored image?



6 × 6 × 3.



3 × 3 × 3

~~4 × 4 × 3~~

4 × 4 × 3

But this is not used.

6 × 6 × 3. 3 × 3 × 3. = 4 × 4 × 1.

↓

This is used.

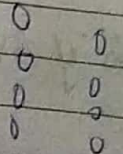
$n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$ → channel (r, g, b)

$p^{[l]}$, $s^{[l]}$, $f^{[l]}$

p → padding in layer l

s → stride in layer l

f → filter size in layer l.



$n_w^{[0]} \times n_H^{[0]} \times n_C^{[0]}$

6 6 3

After convolution :-

$n_w^{[1]} \times n_H^{[1]} \times n_C^{[1]}$

4 4 1

$P^{[1]} = 0$

$S^{[1]} = 1$

$f^{[1]} = 3 \times 3 \times n_C^{[l-1]}$

Why 3×3? → It is nice to have a central element & all rest around it.

$n_w \times n_H \times n_C$ * $3 \times 3 \times 3$ (f_1) → $4 \times 4 \times 1$

6 × 6 × 3 * f_2 → $4 \times 4 \times 1$

* f_3 → $4 \times 4 \times 1$

* f_4 → $4 \times 4 \times 1$

These gives different aspects of the image.

To look keeping stack them on over others.

$4 \times 4 \times 4$

$n_w^{[1]} n_H^{[1]} n_C^{[1]}$

$n_w^{[l-1]} \times n_H^{[l-1]} \times n_C^{[l-1]}$ → $f^{[l]}, P^{[l]}, S^{[l]}$ → # filters

↓ output of last layer

$f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$

4 4 4

$n_w^{[l]} \times n_H^{[l]} \times n_C^{[l]}$

$b^{[l]}$ = [# filters]

linear part is done for conv

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

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

We want to map it here

parameters are filters
take biases by adding.

bias does not depend on filters.

w_1 w_2 w_3
w_4 w_5 w_6 →
w_7 w_8 w_9

We have to have non linear

If we don't have it

$w = e(h(g(f(x))))$

We need non linear in every layers because

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

→ $n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]}$

Relu $(n_w^{[l]} \times n_H^{[l]} \times n_c^{[l]})$

$y < 0$ make 0
$y > 0$ make same.

Every time in each layer we use filters (parameters).

CNN to DNN.

Pooling :-

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

training
validation
test
} Improve accuracy of all.

We do max pooling because we want to extract bright side of an image.

Pooling preserve translation to some extent → Translation invariant.
// does not preserve rotation.
→ you can make Pooling learn the invariant.

In pooling : we keep a 2d filter (imaginary). f×f.

There are no learning parameters.

3×3×3

10 filters

27×10+10=280 parameters.

Camlin Page
Date / /

=# Parameters that you want to learn are invariant to inputs so even if 200x200x3 image, we still have 250 parameters.

But computational cost increases.

But in Deep neural network # parameters are not invariant of input size, that's why we use CNN.