What is activation function?

→ activation function compares input value to a threshold value,

→ If the input value is greater than threshold value, the neuron is activated, it's disable if the input is less than threshold.

→ These functions introduces non-linearity

Common activation functions are

Sigmoid: This function mass input values to a range between 0 and 1.

ReLU (Rectified Linear unit): It replaces negative input values with zero and leaves positive values unchanged.

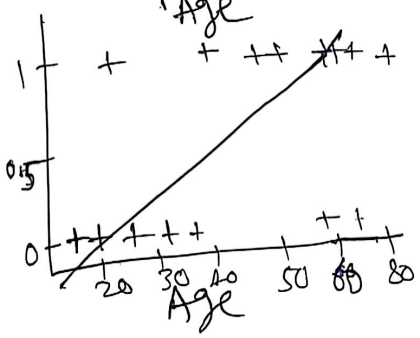
Tanh (hyperbolic Tangent): It squashes input values to a range b/w -1 and 1

$$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$ e = Euler's number ~ 2.71828

Step 1

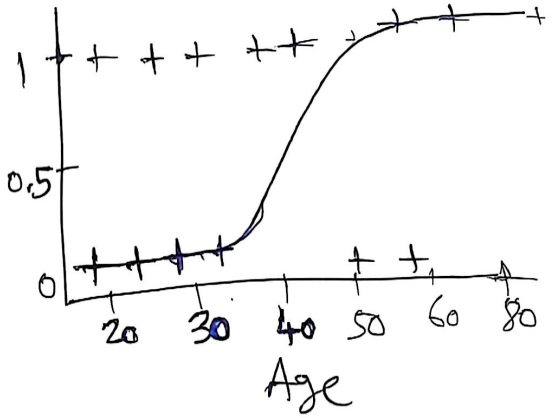
$Y = m*x + b$

Age



Step 2

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



Linear regression:

$Y = 0.042 * X - 1.53$

Age

$Y = 0.042 * X1 + 0.008 * X2 + 0.2 * X3 - 1.53$

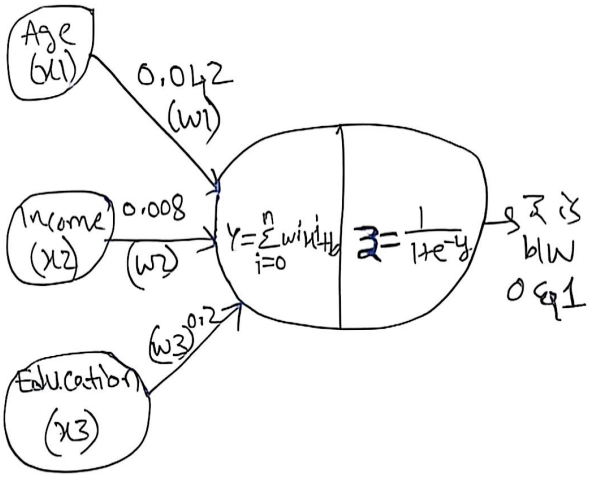
Age, income, education

$Y = W1 * X1 + W2 * X2 + W3 * X3 + b$

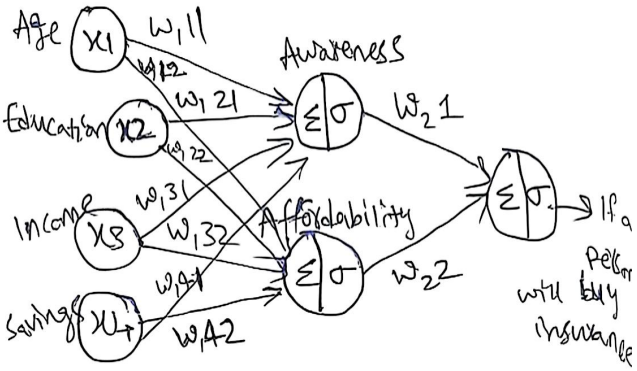
$$y = \sum_{i}^{n} w_i x_i + b$$

$Y = 0.042 * x1 + 0.008 * x2 + 0.2 * x3 - 1.53$

↑ Age ↑ Income ↑ Education

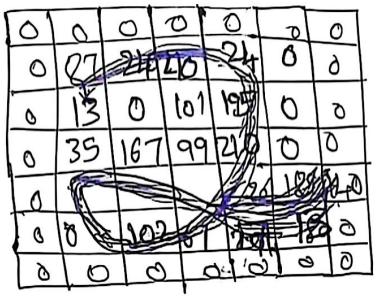


Example of neural network:

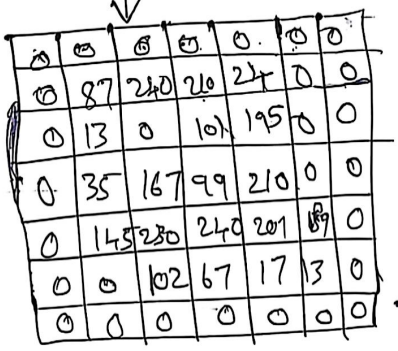


Hand written digits classification.

0 - black 255 - white

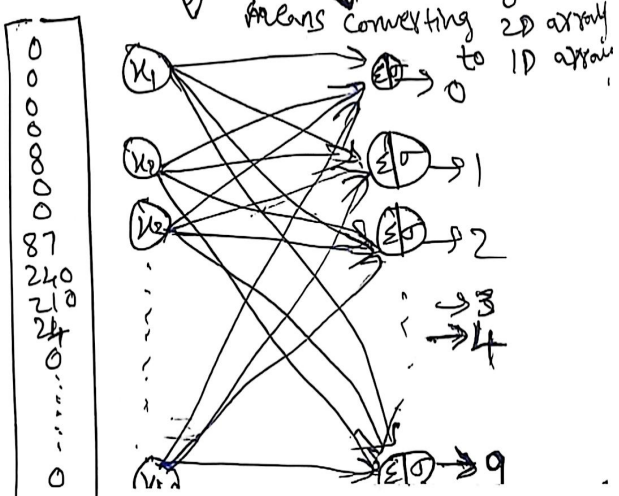


2



7X7 grid

flatten the array means converting 2D array to 1D array



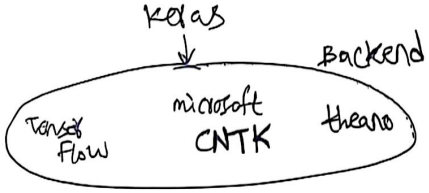
What is a neural network?

→ It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain.

→ neural network comprises an input layer, a hidden layer and an output layer. Deep learning is made up of several hidden layers of neural networks.

Pytorch framework → facebook

Tensorflow " → google



In this we are using

28 x 28 grid

Code:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
```

```
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

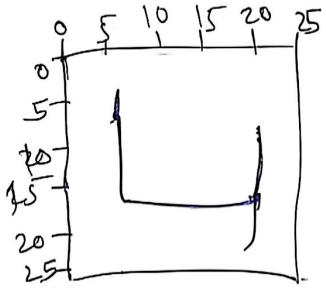
```
P: len(X_train)
op: 60000
```

```
P: X_train[0].shape
op: (28,28)
```

```
P: X_train.shape
op: (60000,28,28)
```

```
P: plt.matshow(X_train[2])
```

op:




```
P: y_train[2]
op: 4
P: y_train[:5]
op: array([5,0,4,1,9], dtype=uint8)
```

```
P: X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_test_flattened = X_test.reshape(len(X_test), 28*28)
P: print(X_train_flattened.shape)
print(X_test_flattened.shape)
op: (60000, 784)
(10000, 784)
```

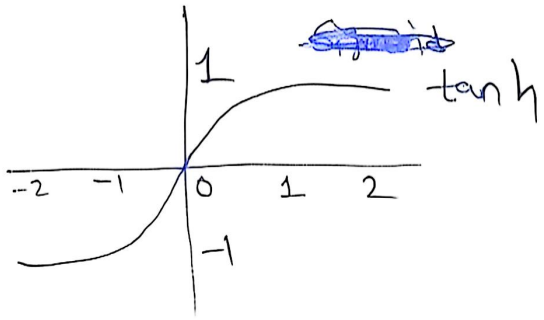
```
P:
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Code is in jupyter notebook.

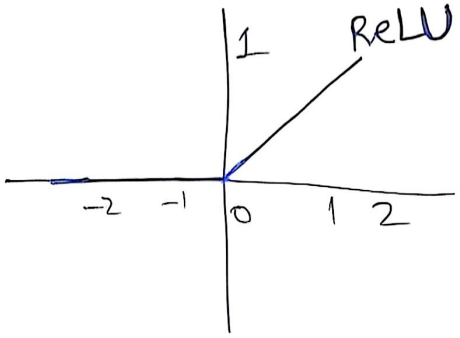
Activation functions

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

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

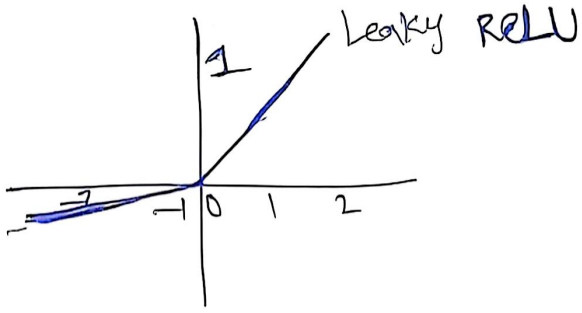


→ use sigmoid in output layer. All other places try to use tanh



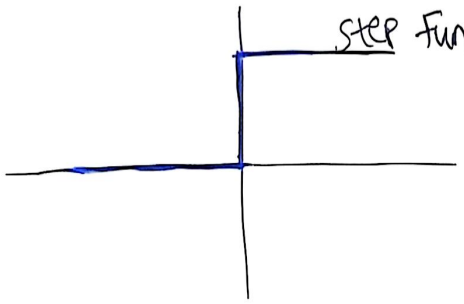
$\text{ReLU}(z) = \max(0, z)$

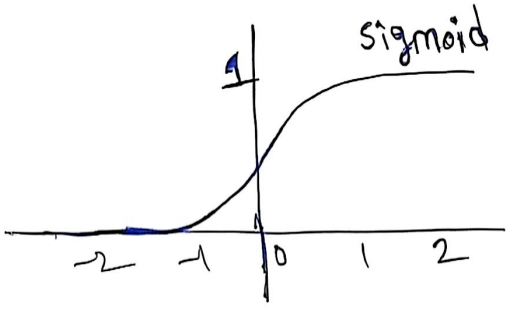
→ For hidden layers if you are not sure which activation function to use, just use ReLU as your default choice.



Leaky $\text{ReLU}(z) = \max(0.1z, z)$

Graphs of activation functions





Slope	Derivative
→ used for linear equation	→ used for non linear equation
→ It is a constant	→ It is a function.

Partial derivatives :-

$f(\text{bedrooms}, \text{sqr ft}) = \text{bedrooms}^3 + \text{sqr ft}^2$

$\frac{\partial(\text{price})}{\partial(\text{bedrooms})} = 3 * \text{bedrooms}^2$

How much a price is changing given a change in bedrooms.

$$\frac{\partial(\text{price})}{\partial(\text{sq ft})} = 2 * \text{sq ft}$$

How much a price is changing given a change in sq ft

Tensorflow loss value examples:

- Sparse_categorical_Crossentropy
- binary_crossentropy
- Categorical_crossentropy
- mean_absolute_error
- mean_squared_error

if you have 13 rows of insurance data

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
.		
.		
.		
28	1	0
27	0	1

$Y = w1 * x1 + w2 * x2 + bias$

$x1$: age, $x2$: affordability

example:

$\text{error}_1 = \text{abs}(y - \hat{y}) = 0.99$

$\text{error}_2 = \text{abs}(y - \hat{y}) = 0.99$

$\text{error}_{13} = \text{abs}(y - \hat{y}) = 0.01$

Mean Absolute error = $9/3 = 3$

mean squared error = $25+9+1 = 35/3$

Total error = $\text{error}_1 + \text{error}_2 + \cdots + \text{error}_{13}$ ← loss

$$= \sum_{i=1}^{n} \text{abs}(y_i - \hat{y}_i)$$

→ individual errors are called 'loss'

mean Absolute Error (MAE) = $\frac{1}{n} \sum_{i=1}^{n} \text{abs}(y_i - \hat{y}_i)$ ← Cost function

→ Mean absolute error is called Cost function.

Mean Squared Error (MSE) $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

log loss or binary cross entropy

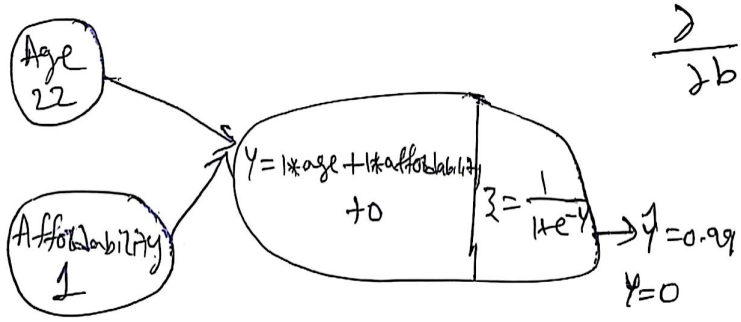
$$= -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1-y_i)\cdot\log(1-\hat{y}_i)$$

Gradient descent :-

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
.		
.		
.		
28	1	0
27	0	1

$y = w1 * x1 + w2 * x2 + bias$

x1 ↑ Age, x2 ↑ Affordability



$error_1 = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$

$= 4.6$

$error_2 = 4.6$

$error_{13} = 0.01$

for first epoch

Total error $= error_1 + \dots + error_{13}$

log loss or binary cross entropy $= \frac{-1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$

After first epoch

log loss = 4.31

→ Back propagate this loss to adjust weights to reduce this log loss.

For adjusting weights

$w1 = w1 - \text{something}$

$w1 = w1 - \text{learning rate} * \partial/\partial w1$

↓

→ gt is basically 0.01

→ gt is limiting derivative function.

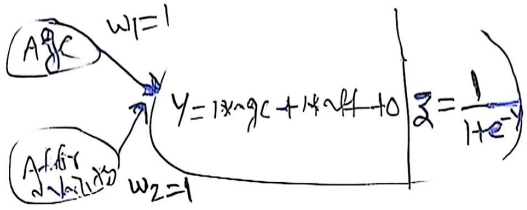
$w2 = w2 - \text{learning rate} * \partial/\partial w2$

$b = b - \text{learning rate} * \partial/\partial b$

$$\frac{\partial}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} x_i (\hat{y}_i - y_i)$$

$$\frac{\partial}{\partial w_2} = \frac{1}{n}\sum_{i=1}^{n} x_i (\hat{y}_i - y_i)$$

$$\frac{\partial}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)$$



After first epoch

log loss = 1.31

And weights are adjusted

$w1 = w1 - \text{learning rate} * \partial/\partial w1$

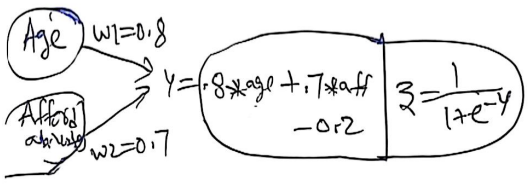
$w1 = 1 - 0.2 = 0.8$

$w2 = w2 - \text{learning rate} * \partial/\partial w2$

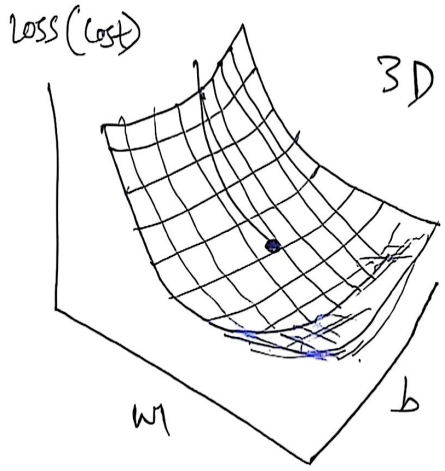
$w2 = 1 - 0.3 = 0.7$

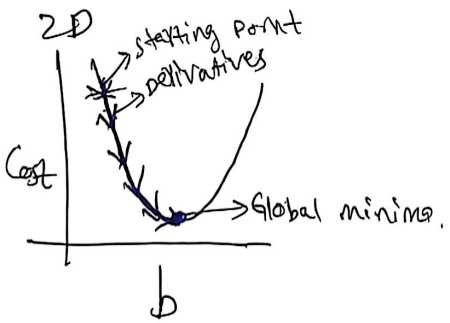
$b = b - \text{learning rate} * \partial/\partial b$

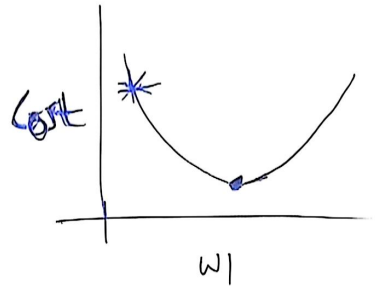
$b = 0 - 0.2 = -0.2$



Then feed the samples and find the error for all samples. ie second epoch







Batch Gradient Descent:

→ use all training samples for one forward pass and then adjust weights

→ good for small datasets.

Stochastic Gradient Descent (SGD)

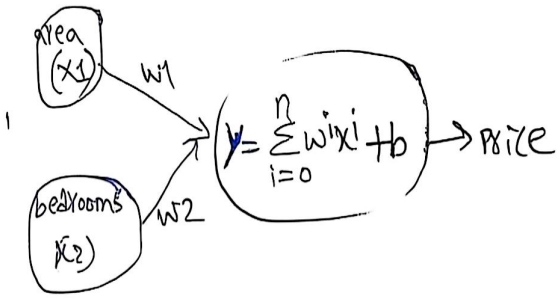
→ use one (randomly picked) sample for a forward pass and then adjust weights.

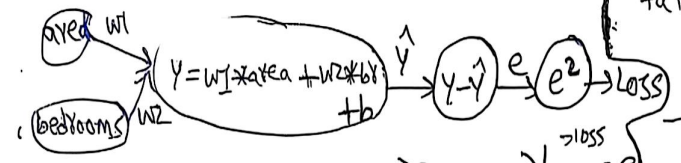
→ Best for large datasets.

Mini batch gradient descent:

→ use a batch of (randomly picked) samples for a forward pass and then adjust weights.

Chain rule!





$\frac{\partial \hat{y}}{\partial w_1} = area$

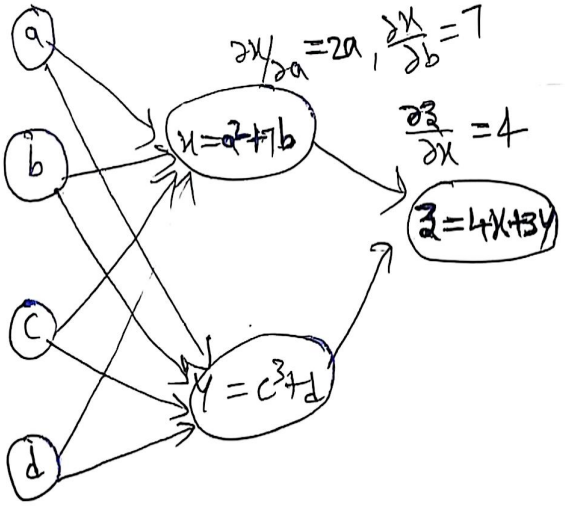
$\frac{\partial e}{\partial \hat{y}} = -1 \quad \frac{\partial L}{\partial e} = 2e$ (loss)

$\frac{\partial L}{\partial w_1} = ?$ (chain rule)

$\Rightarrow \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial e} * \frac{\partial e}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$

$= -2e * area$

$\frac{\partial L}{\partial w_1} = -2e * area$



$\frac{\partial z}{\partial a} = \frac{\partial z}{\partial x} * \frac{\partial x}{\partial a} = 4*2a = 8a$

$\frac{\partial z}{\partial b} = \frac{\partial z}{\partial x} * \frac{\partial x}{\partial b} = 4*7 = 28$

	1	2	3	4	5	6	7	8	9	10
Truth:	D	C	D	D	D	P	H	D	D	B
Prediction:	D	D	D	ND	D	ND	D	ND	D	D

D→Dog, ND→NO Dog, P→Phone, C→Car, B→Bird

True Positive = 4 (1,3,5,9)

False Positive = 3 (2,7,10)

True Negative = 1 (6)

False Negative = 2 (4,8)

How many we got right? → 5

Accuracy → 5/10 → 0.5

For dog-class -

→ Precision is out of all dog predictions how many you got it right?

Precision = 4/7 = 0.57

Precision = TP / (TP+FP)

→ Recall is out of all dog truth how many you got it right?

Total Dog truth samples. = 6 (1,3,5,4,8,9)

True Positive = 4

6 dog truth samples ఉంటే దానిలో నువ్వు ఎంతమంది కరెక్ట్ గా predict చేశావు.

4/6 = 0.67

$Recall = \frac{TP}{(TP+FN)}$

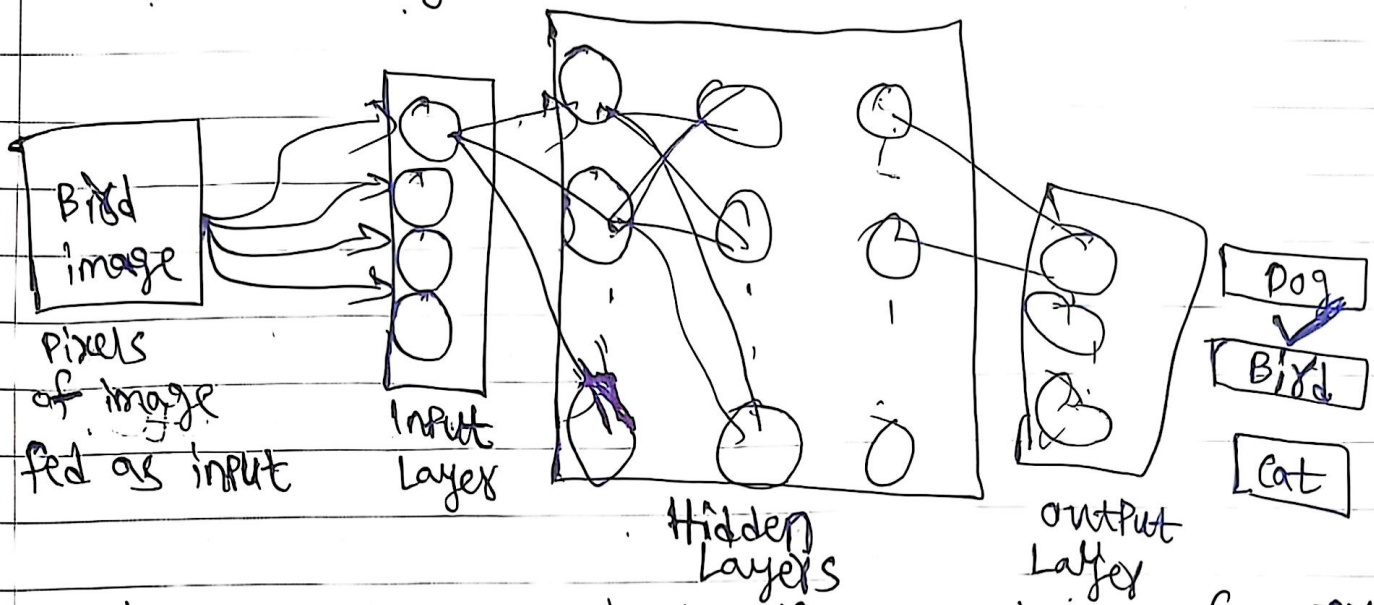
→ For Precision, think about predictions as your base.

→ For recall, think about truth as your base.

4/19/2024

Convolution Neural Network

How image recognition works?
→ It does using a Convolution Neural Network



Let's see how CNN identifies the image of a Bird.

→ "Input Layer" accepts the pixels of the image as input in the form of arrays.

→ "Hidden Layers" carry out feature extraction by performing certain calculation and manipulation.

Convolution Layer
→ The layer after the input Layer.
→ This layer uses a matrix filter and performs convolution operation to detect patterns in the image.

Matrix filter

1	0	1
0	1	0
1	0	1

Note:
* There are multiple hidden layers like Convolution layer, ReLU layer, Pooling layer etc... that perform feature extraction from the image.

Rectified Linear Unit : (ReLU)

→ ReLU activation function is applied to the convolution layer to get a rectified feature map of the image.

Pooling :

→ Pooling layer also uses multiple filters to detect edges, corners, eyes, feathers, beak etc...

→ Finally there is a fully connected layer that identifies the object in the image.

What's in it for you?

→ Introduction to CNN
→ What is convolution neural network?
→ How CNN recognizes images?
→ Layers in CNN
→ Use case implementation using CNN.

Intro to CNN :

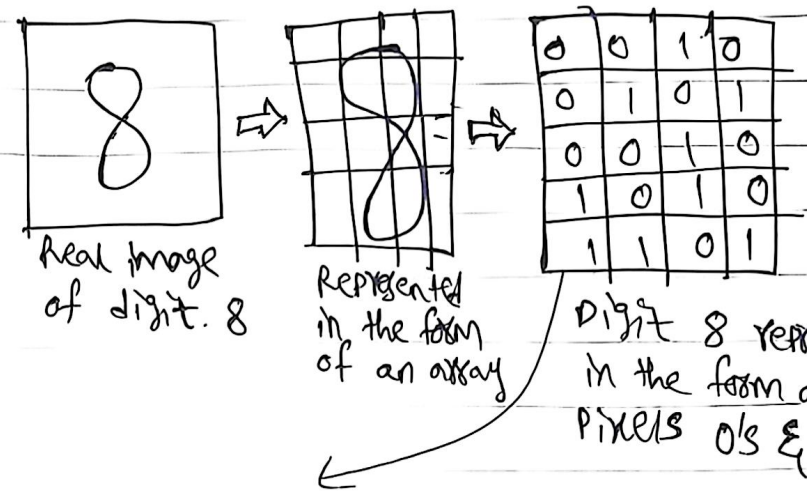
→ Pioneer of CNN - "Yann LeCun"
→ Built the first CNN called LeNet in 1988
→ It was used for character recognition tasks like reading ZIP codes, digits.

What is a CNN?

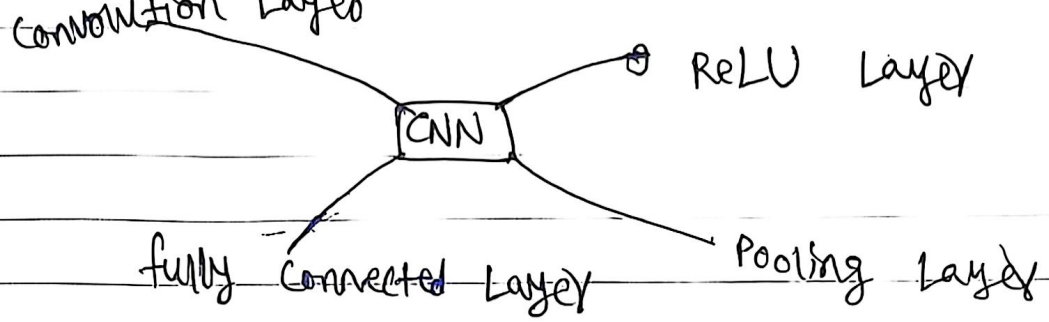
→ CNN is a feed forward neural network that is generally used to analyze visual images by processing data with grid like topology.

Convolution operation forms the basis of any Convolution Neural Network.

In CNN, every image is represented in the form of arrays of pixel values



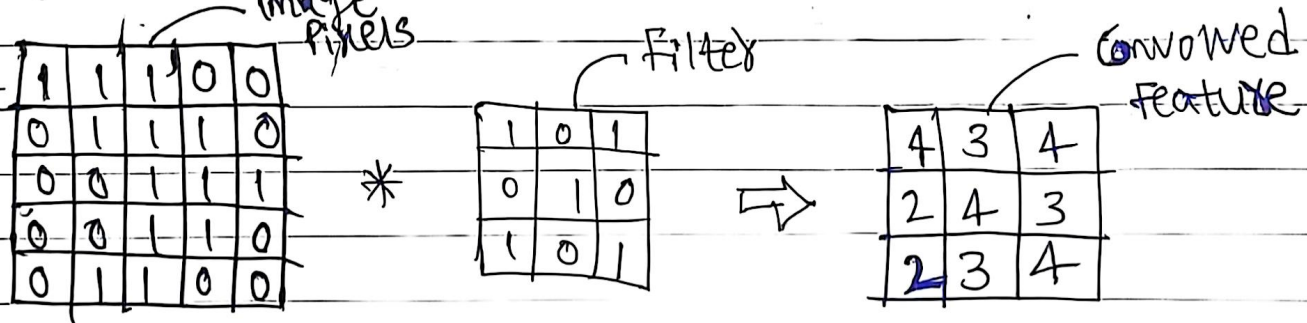
Layers in ~~CNN~~ CNN



Convolution Layer

→ It has a number of filters that perform convolution operation.

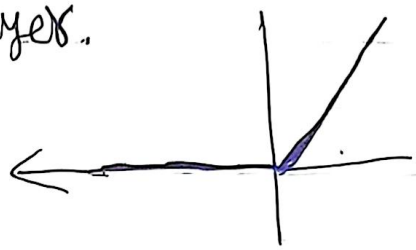
→ Every image is considered as a matrix of pixel values.



sliding the filter matrix over the image and

ReLU:

Once the feature maps are extracted, the next step is to move them to a ReLU layer.



$R(z) = \max(0, z)$

ReLU

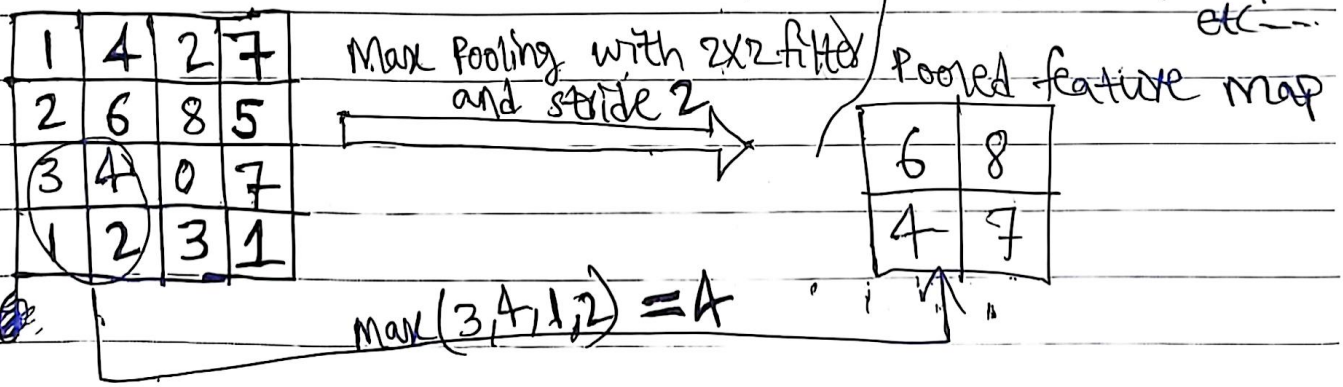
→ performs element wise operation
→ sets all negative pixels to 0.
→ Introduces non-linearity to the network.
→ The output is a rectified feature map.

Pooling Layer:

→ The rectified feature map now goes through a pooling layer.

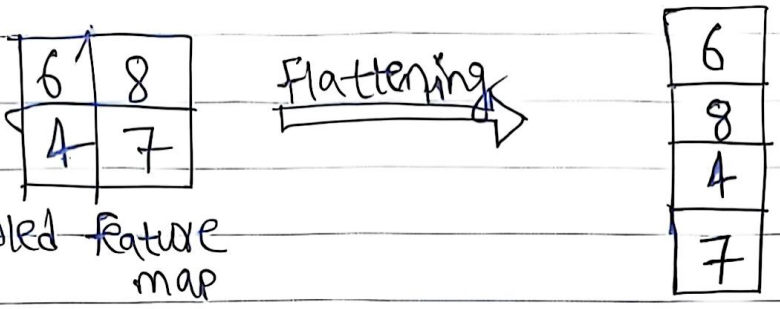
* → Pooling is a down-sampling operation that reduces the dimensionality of the feature map.

* → Pooling layer uses different filters to identify different parts of the image like edges, corners, body, feathers, eyes, beak etc...



Flattening

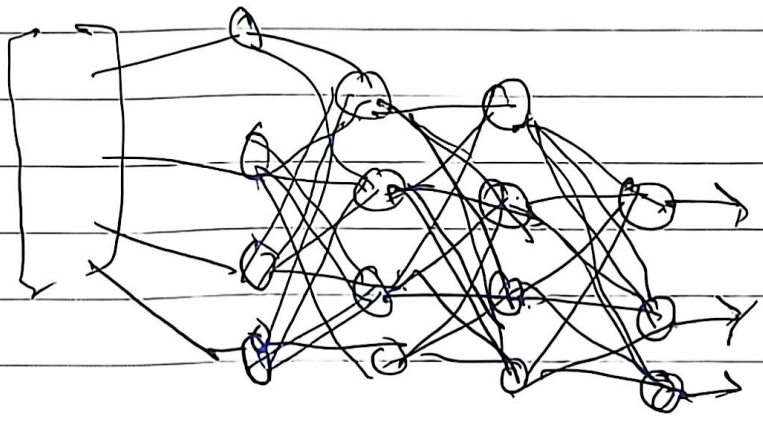
→ It is the process of converting all the resultant 2 dimensional arrays from pooled feature map into a single long continuous linear vector.

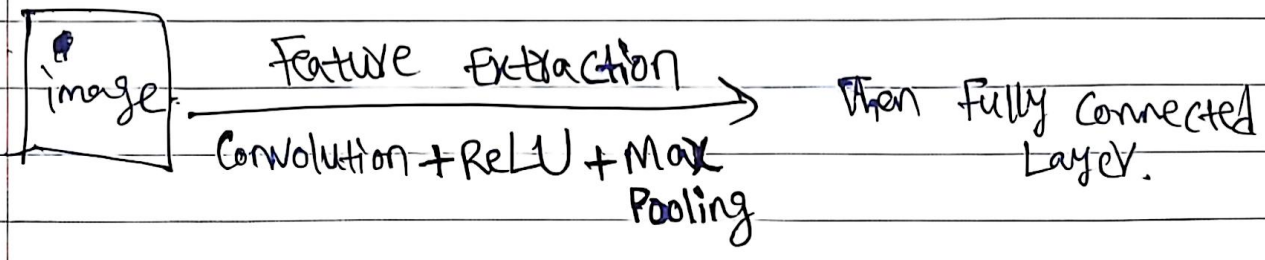


Pooled feature map

Fully Connected Layer:

→ The flattened matrix from the pooling layer is fed as input to the Fully Connected Layer to classify the image.





Use case implementation using CNN :-

→ In this we will be using CIFAR-10 data set (from Canadian Institute for Advanced Research) for classifying images across 10 categories.

01 - Aeroplane	06 - dog
02 - Automobile	07 - Frog
03 - bird	08 - horse
04 - Cat	09 - ship
05 - deer	10 - truck