04/08/25

Experiment No-01

Basic Python Codes

1) find the square root of number

```
import math
n = int(input())
print(math.sqrt(n))
```

O/P:-

```
4
2.0
```

2) Swap two numbers

```
a = int (input ("enter 1st num:"))
b = int (input ("enter 2nd num:"))

c = a
a = b
b = c
print (a)
print (b)
```

O/P:-

```
enter 1st num = 2
enter 2nd num = 3
3
2
```

3) checking num is positive (or) not

```
n = int (input())
if (n>=0);
print ("num is positive")
else:
print("num is negative")
```

o/p:-

4

num is Positive

4) check num is even or odd

```
n = int (input())
if (n%2 == 0):
print ("num is even")
else:
print ("num is odd")
```

o/p:-

5

num is odd

5) Find the factorial of num

```
import math
n=int (input())
print (math. factorial (n))
```

o/p:- 3

6

6) check whether the number is Palindrome

```
n = inPut()
if (n == n [::-1]);
print ("Palindrome")
else:
print ("not a Palindrome")
```

o/p:- 323

Palindrome

7) use Pandas & create a dataset, print age & age>25

```
import Pandas as Pd
data = {"Name": ["chaxan", "Raju", "Suxya"],
        "Age" : [20,21,22]
        "city" : ["hyd", "hyd", "Bgxl"]}
df = Pd. Data fxame (data)
print(df)
```

o/p:-

	Name	Age	city
0	chaxan	20	hyd
1	Raju	21	hyd
2	suxya	22	Bgxl

```
df ["Age"]
```

o/p:-

	Age
0	20
1	21
2	22

```
df[df ["Age"] >25]
```

o/p:-

---

8) Implemented AND gate with 3 inputs

```
def AND_gate(a,b,c):
    return a and b and c

inputs = [(0,0,0),
          (0,0,1),
          (0,1,0),
          (0,1,1),
          (1,0,0),
          (1,0,1),
          (1,1,0),
          (1,1,1)]

print("ABC | output")
print("-------")
for a,b,c in inputs:
print(f"{a} {b} {c} {int(AND_gate(a,b,c))}")
```

O/P

A	B	C	O/P
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

O/P verified

Experiment-2

```
from transformers import pipeline
sentiment_analyzer = pipeline("sentiment analysis")
text=" I am feeling happy"
sentiment_result = sentiment_analyzer(text)
print("sentiment analysis:")
print(f" sentiment : {sentiment_result[0]['label']}
      (score: {sentiment_result[0]['score']:.2f
```

O/P:-

Sentiment Analysis
Sentiment : POSITIVE (score: 1.00)

```
from transformer import pipeline
translator_en_ta_fr = pipeline ('translation_in_to_fr',
                                model = "Helsinki-NLP/opus-int-en-to")

text = "I am feeling happy"
translation_result = translator_in_to_fr(text)
print("Translation:")
print(f" Translated text = {translation_result[0]
                            ['translation_text \n")
```

O/P:-

Translated text: Je ke sen heuruse.

Exp:- 2(b)

```
! pip install pytesseract
! pip install pillow
! pip install transformers

import pytesseract
from PIL import Image
from transformers import

input = "/content/image.png"
input_image = Image.open(input)
text_output = pytesseract.image_to_string(input_image).
  strip()
print(text_output)

from transformers import pipeline
Sentiment analyzer = pipeline("sentiment-analysis")
text = text_out put
Sentiment_result = Sentiment_analyzer(text)

print("sentiment analysis:")
print(f"Text: {text}")
print(f"Sentiment: {sentiment_result[0]['label']}
  Score: {Sentiment_result[0]['score']:.2f}")
```

O/P:- Sentiment analysis:

Text: It was the best of times,
It was the worst of times,
It was the age
Sentiment: NEGATIVE Score: 0.98

O/P 1/3/8/20

Experiment No-03

Variational Auto Encoder

```
import numpy as np
import Tensorflow as tf
 import Keras
from Keras import Layers
```

1. Define Sampling Layer
   -> Take mean & log variance from encoder output
   -> Sample z using epsilon * Exp(0.5 * log-var) + mean

2. Build Encoder
   -> input (28,28,1) image
   -> Apply Conv2D layers to Extract features
   -> flatten & use dense layers
   -> o/p :- mean, log-var & sampled latent vector z

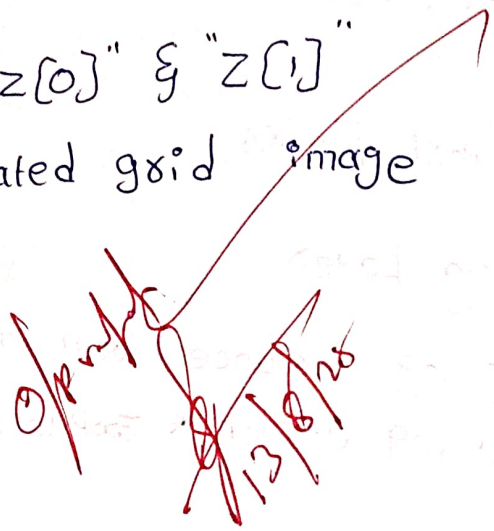
3. Build Decoder
   -> input : latent vector (latent_dim)
   -> Dense - reshape to (7,7,64)
   -> Conv2D Transpose layers to upsample back to (28,28,1) image

4. Define VAE model
   -> Combine encoder decoder
   -> Train the model apply gradient to update weights

5. Train VAE
   -> load & normalize fashion-MNIST data
   -> fit model for desired epoches.

6. Plot latent space

-> create a 2D grid of latent points
-> for each (xi, yi), decode to generate an image
-> place image in correct spot in large figure grid
-> label axes as "z[0]" & "z[1]"
-> Show the generated grid image



Experiment No-04

Generative Adversarial Network (GAN)

1) Load & Prepare Data

Download Data set ex:- MNIST
Normalize/Scale images
Add a channel dimension
Create a batched dataset for training

2) Build the Generator

Input: Random noise vector
Use Dense + Reshape to create initial feature map
Apply Conv2D Transponse layers
Out Put :- Image with shape matching MNIST (28x28 x1)

3) Build the Discriminator

Input: Real or generated image
use conv2D layers to downsample & extract featu
Apply Dense + Sigmoid activation for binary clasficatio

4) Set up Loss and optimizers

use Binary Cross-Entropy Loss
Define Generator Loss
Define Discriminator Loss
use Adam optimizers

5) Define the Training step.

Generate fake images from noise using t
generator
Pass both real & fake images to the discrimi
Compute generator & discriminator losses
Calculate gradients & update weights using opti

6) Run the Training Loop

Iterate through epoches & dataset batches
Call the training step for each batch
Periodically generate & save sample images to
monitor progress

o/p output

Experiment No-05

Image generation

1) Install dependencies

```
! pip install diffesers accelerate transformers safetensors
  torch torchvision --upgrade
```

Import libraries

```
import torch
from diffusers import PixArtAlphaPipeline
```

2) Load a Transformer

```
model_id = "PixArt-alpha/PixArt-XL-2-512x512"
dtype = torch.float16 if torch.cuda.is_available() else
  torch.float32
pipe = PixArtAlphaPipeline.from_pretrained(model_id,
        torch_dtype = dtype)
device = "cuda" if torch.cuda.is_available else "cpu"
pipe = pipe.to(device)

  if device == "cuda":
        pipe.enable_vae_slicing()
```

3) Promt for face

```
Promt = ("ultra-realistic closeup portrait of a young
      adult human, natural expression."
      "natural soft lighting, 85mm lens boken, high
      detail skin, Photographic quality")
```

4) Generate image

```
generator = torch.Generator(device=device).manual
```

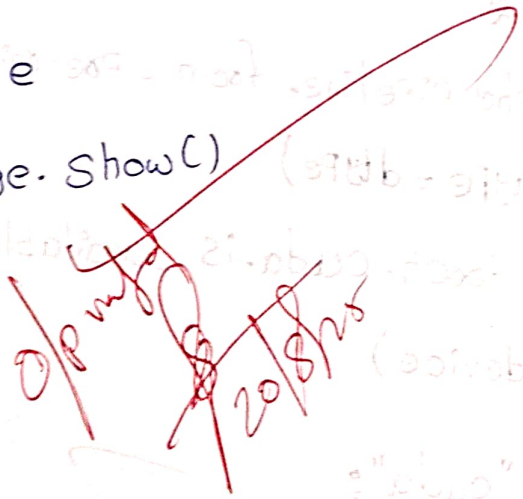
```
image = pipe(
    prompt = prompt,
    negative_prompt = negative_prompt,
    num_run = 24
    height = 512
    width = 512
    generator = generator).images[0]
```

5) Save & display

```
image.save("face.png")
print("saved to face.png")
```

6) Show image

```
image.show()
```



Experiment No-07

Conditional GAN

1) Initialization

-> Set hyperparameters: batch size, learning rate, latent vector size, number of epoches (9)

-> Initalize Generator G and Discriminator D

-> Define loss function and optimizers

2) Data preperation

-> Load MNIST Dataset

-> Normalize images to the range [-1,1]

-> Convert labels into embeddings for Conditioning

3) Training Loop

a) for each batch

-> Generate noise vector z & sample labels

-> Generate fake image G(z,y)

-> Train Discriminator with Real & fake

b) update Parameters

-> Back propogate & update D

-> Backpropogate & update G

4) observation Per Epoch

-> Record Discriminator loss & Generator loss

-> observe training stability (D-loss vs G-loss)

5) Visualization

→ Every 5 epoches, generate digits for labels 0-9

→ compare the clarity & quality of digits across epoches

→ Note improvements in shape, sharpness & diversity

6) Final observation:-

→ After 9 epoches, evaluate how realistic the generated digits are

→ observe whether the Generator produces label-conditioned digits correctly.

Pseudo Code:-

```
Initialize G, D, loss, optimizers
Load MNIST Dataset

For epoche = 1 to 9
    for each batch (X, Y):
        z ← random noise
        fake ← G(z, Y)
        D_loss ← loss(D(X, Y), 1) + loss(D(fake, Y), 0)
        update D

        z ← random noise
        fake ← G(z, Y)
        G_loss ← loss(D(fake, Y), 1)
```

Update G

END

Print (epoch, D:loss, G_loss)

If epoch % 5 == 0: show generated digits

END

o/p verified

1/9/25

Experiment No-08

Retrieval - Augmented Generation

Input:-

-> A set of structured & unstructured document
-> a user query
-> pretrained embedding model
-> pretrained generation model

output:-

-> a generated answer to the query based on retri-eved documents.

Algorithm:-

1. preprocessing documents:-

- Convert all structured document into a textual format that can be embedded
- Keep unstructured document as they are

2. embedding documents

- use a pretrained embedding model to compute vector embeddings for all documents
- Store these doc embeddings in the vector index structure for fast similarity search

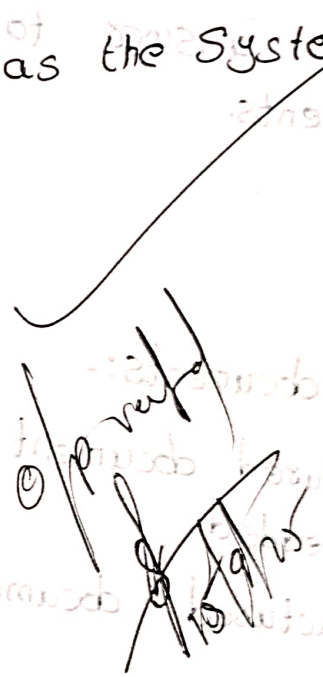
3. Query processing & Retrieval

- Embed the incoming user query using the same Embedd-ing model
- Perform Similarity search on the vector index with query embedding to retrieve the top-K most relevant documents.

4. Answer Generation

- Concatenate the retrieved documents into a single textual content
- Construct an input prompt for the generative model by Combining the query and retrieved contextly
- use a pretrained generative Language model to generate a natural language answer unconditional on prompt

5. Return the generated answer as the System's response to query.



Experiment:- 09

Customizing LLM on Local Machine

Algorithm:-

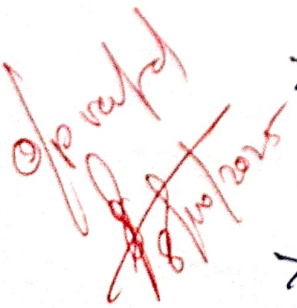
1) Install Anaconda mini in our system
2) open Conda prompt

Prompts:-

-> conda env list
-> condacreate --- name genai python == 3.9
-> conda activate gen ai
3) install the olla usa
4) version ollauxa 3.2
5) open canuael prompt

promte:-

> Ollanura vun Uaura 3.2
>>> what is capital of India.
The capital of India is new delhi
> from / Uicune -336.Q4 -Q.99uf
>Ollana create example f newlife
>echo. > Moclelfile
>olllaya create Maiu.f./mocleltile
>> Ollama vun mario
>> hi
its a Me, Mario! How can I help to u



Experiment No:-10

Gradio APP (Customizing LLMs)

1) Install Gradio

Use !pip install gradio to make sure the library is available

2) Install & Import Gradio library

3) Define the function that processes input & produces output

4) create Interface

wrap the function with a Gradio Interface or Blocks to connect inputs & outputs.

5) Gradio interface by linking

- function (fn=.....)
- Input component (inputs=.....)
- output component (outputs=.....)

6) Launch the interface using .launch()

7) choose Input/output widgets.

Textbox → for text input

Slider → for numeric range

Checkbox → for true/false

Radio/Drop down → for multiple-choice options

Image/video/Audio → for multimedia input

JSON → for structured data

Blocks + Rows/Columns → for custom layouts

8) Launch the APP:-

Call .launch() to open a local Gradio app in browser.

O/P verified

15/9/25

Github link verified for all first ten experiments

8/10/2025

Experiment -11(a)

Set up a fast API chat App

Algorithm:-

1) Install required Packages: fast API. uvicorn nest asyncio. Pyngrok
2) Import necessary modules
3) Kill existing ngrok modules
4) Apply nest asyncio Patch
5) Create a Fast API app
6) Define a message data model
7) Create API end points
8) Kill any existing ngrok tunnels.
9) Start ngrok tunnel on Port 8000
10) Set ngrok auth token
11) Configure & run Uvicorn Server
12) Run the Uvicorn Server asynchronsely inside the Current event loop.

Experiment - 11(b)

Fast API Chat Application with ngrok tunnel for Public access in Colab.

Algorithm:-

1) Install required Packages (fast API, Uvicorn, nest asyncio, Pyngrok)
2) Kill any existing ngrok process & wait for termination
3) Apply nest_asyncio to allow nested async event loops in colab
4) Create a Fast API app with endpoints to send & retrieve chat messages stored in memory
5) Kill any previous ngrok tunnels for a clean start
6) Set ngrok authentication token
7) Start an ngrok tunnel exposing Port-8000 & print the public url.
8) Configure & run the Uvicorn server inside the current async loop to serve the fast API app

Output

Experiment No:-12

Training an RL Agent to Navigate a Taxi using Q-Learning & LLM Integration

Problem Statement:-

-> Train a RL agent using Q-Learning to navigate a taxi in a grid world

-> The agent must pick up a passanger from one location and drop them off at a dection while minimizing steps and avoiding illegal actions

-> An LLM is integreated to Explain Police & suggs Parameter

Algorithm:-

1) Initialize:-

Create a table with all zero for state-action Pairs

2) Set HyperParameters:-

Learning rate (α), Discount factor (γ^c) Explo-ration rate (ϵ) & episides.

3) for each episode:-

- Reset environment -> get initial state
- Repeat until goal reached:

a - choose an action using ϵ-greedy policy

b. Perform action -> observe next state &

γ ecaured

c. update q-value using

$$Q(S,\alpha) = Q(S,\alpha) + \alpha[\gamma + \gamma \max Q)(s',a') - Q(S,\alpha)$$

d. set next state as current state

29/10/2025

t