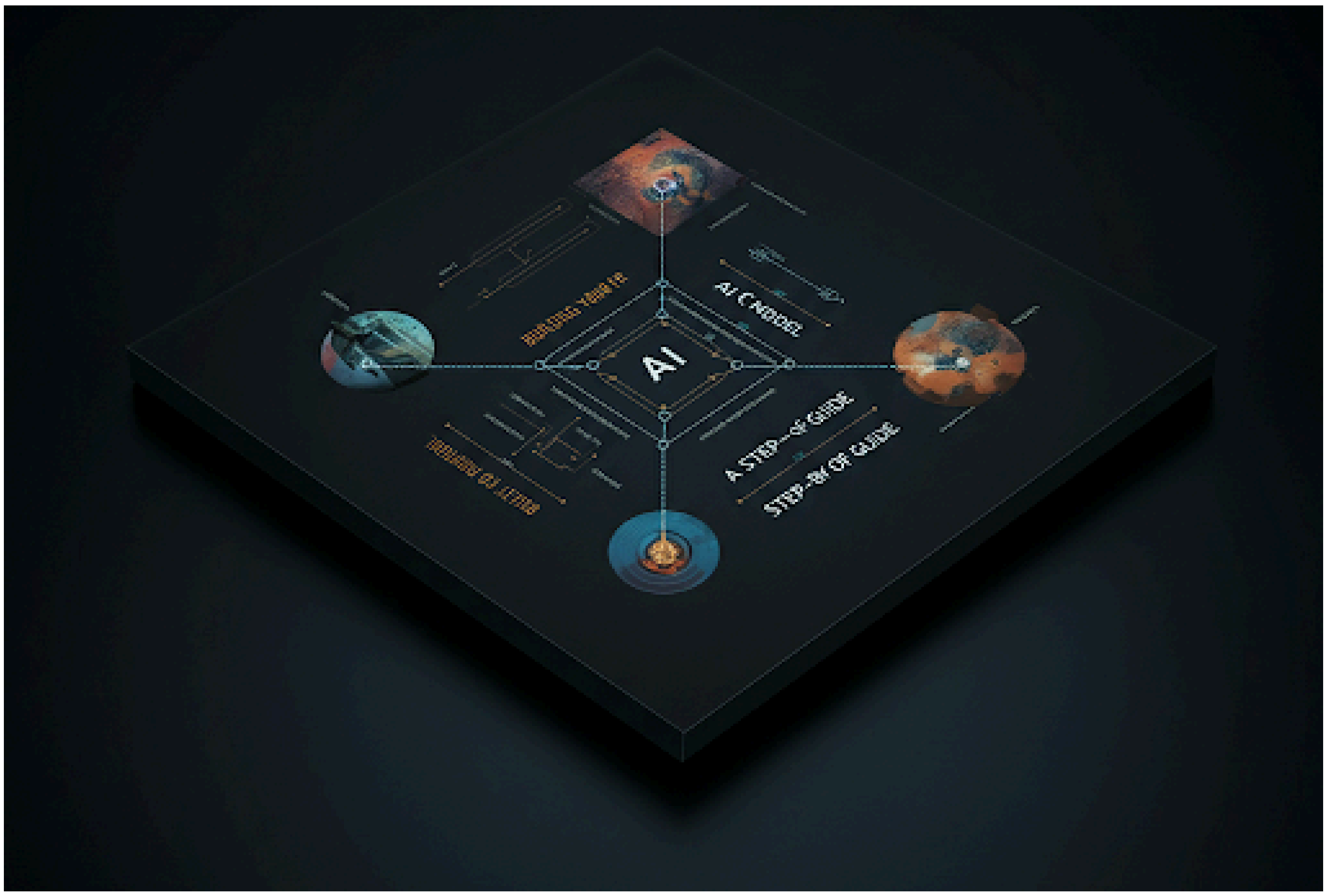
POST 45- GEN AI

GENERATIVE AI
FOR ALL

Your First Generative AI Model

A Step by Step Guide



DINESHLAL

Introduction

- Generative AI is a rapidly growing field that allows computers to create new content like text, images, and music, much like humans do.
- This guide offers a beginner-friendly, step-by-step tutorial to build your first generative AI model.
- We'll use popular libraries such as TensorFlow, Keras, and PyTorch. By the end, you'll grasp the basics and have built a simple model.

Key Generative AI Models

- **Generative Adversarial Networks (GANs):** These are the focus of our tutorial. They work by having two neural networks compete: one generates content, the other judges its quality.
- **Variational Autoencoders (VAEs):** These models learn the underlying structure of data to generate new, similar data.
- **Transformers:** Primarily used for text and sequences, these models are behind powerful language models like GPT-3.

Prerequisites

- **Programming Skills:** Basic understanding of Python is essential.
- **Frameworks and Libraries:**
  - **TensorFlow/Keras (or PyTorch)** - Deep learning frameworks.
  - **NumPy** - For numerical operations.
  - **Matplotlib** - For data visualization.

Prerequisites

- **Development Environment:**
  - Python 3.8+ installed.
  - Jupyter Notebook or an IDE (e.g., PyCharm, VS Code).
  - Install necessary libraries using pip: pip install tensorflow numpy matplotlib
- **Dataset:**
  - We'll use the MNIST dataset, a classic collection of handwritten digits (0-9). Think of it as the "Hello, World!" of image data.

Step-by-Step Guide to Building Your First GAN

1. **Understanding GANs**

- A GAN has two main parts:
    - **Generator**: This creates new data instances, like images, that resemble the training data. Imagine it as an art forger trying to create a masterpiece.
    - **Discriminator**: This acts like an art expert, trying to distinguish between real images from the training set and fake ones created by the generator.

Step-by-Step Guide to Building Your First GAN

1. **Understanding GANs**
    - These two networks are in a constant battle. The generator tries to fool the discriminator, while the discriminator tries to catch the fakes. This "adversarial training" is what makes GANs so powerful.

Step-by-Step Guide to Building Your First GAN

2. Import Required Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import
Dense, Reshape, Flatten, Dropout
from tensorflow.keras.models import
Sequential
import numpy as np
import matplotlib.pyplot as plt
```

Step-by-Step Guide to Building Your First GAN

3. Load the Dataset

This code loads the MNIST dataset and preprocesses it. Normalizing the pixel values to a range of -1 to 1 helps the GAN train more effectively.

```
# Load MNIST dataset

(x_train, _), (_, _) =
tf.keras.datasets.mnist.load_data()

# Normalize images to [-1, 1] range for
better GAN performance
x_train = x_train / 127.5 - 1.0 x_train =
x_train.reshape(-1, 28, 28, 1)
print(f"Dataset shape: {x_train.shape}")
```

Step-by-Step Guide to Building Your First GAN

4. Build the Generator

This code defines the generator network. It takes a random noise vector (of size 100) as input and uses dense layers with ReLU activation to transform it into a 28x28 image. The tanh activation ensures the output is in the desired range.

```
def build_generator():
 model = Sequential()
 model.add(Dense(128, input_shape=(100,),
activation='relu'))
 model.add(Dense(784, activation='tanh'))
return model

# Output size = 28x28=784
model.add(Reshape((28, 28, 1))) return
model generator = build_generator()
generator.summary()
```

Step-by-Step Guide to Building Your First GAN

5. Build the Discriminator

This code defines the discriminator. It takes a 28x28 image as input, flattens it, and uses dense layers to classify it as real (output close to 1) or fake (output close to 0).

```
def build_discriminator():
 model = Sequential()
 model.add(Flatten(input_shape=(28, 28,
1)))
 model.add(Dense(128, activation='relu'))
 model.add(Dense(1,
activation='sigmoid'))
# Output: Real (1) or Fake (0)
 return model

discriminator = build_discriminator()
discriminator.summary()
```

Step-by-Step Guide to Building Your First GAN

6. Compile the GAN Components

Here, we compile the discriminator with the binary_crossentropy loss function and the adam optimizer. We also create the combined GAN model, where the generator's output is fed into the discriminator.

```
# Compile the discriminator

discriminator.compile(loss='binary_crosse
ntropy', optimizer='adam',
metrics=['accuracy'])

discriminator.trainable = False # Freeze
discriminator during generator training
```

Step-by-Step Guide to Building Your First GAN

6. Compile the GAN Components

Here, we compile the discriminator with the binary_crossentropy loss function and the adam optimizer. We also create the combined GAN model, where the generator's output is fed into the discriminator.

```
# Build the GAN model
def build_gan(generator, discriminator):
 model = Sequential()
 model.add(generator)
 model.add(discriminator)
 return model

gan = build_gan(generator, discriminator)
gan.compile(loss='binary_crossentropy',
optimizer='adam')
```

Step-by-Step Guide to Building Your First GAN

7. Train the GAN

```
def train_gan(generator, discriminator, gan, data,
epochs=10000, batch_size=128):
    half_batch = batch_size // 2

    for epoch in range(epochs):
        # Train Discriminator
        idx = np.random.randint(0, data.shape[0],
half_batch)
        real_images = data[idx]
        noise = np.random.normal(0, 1, (half_batch, 100))
        fake_images = generator.predict(noise)

        d_loss_real =
discriminator.train_on_batch(real_images,
np.ones((half_batch, 1)))
        d_loss_fake =
discriminator.train_on_batch(fake_images,
np.zeros((half_batch, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Train Generator
        noise = np.random.normal(0, 1, (batch_size, 100))
        g_loss = gan.train_on_batch(noise,
np.ones((batch_size, 1)))

        # Print progress
        if epoch % 1000 == 0:
            print (f"Epoch {epoch} | D Loss: {d_loss[0]} |
G Loss: {g_loss}")
            plot_generated_images(generator)
```

Step-by-Step Guide to Building Your First GAN

7. Train the GAN

This is the core training loop. In each epoch, we:

- Train the discriminator on a batch of real and fake images.
- Train the generator to fool the discriminator.
- Periodically print the progress and visualize generated images.

```
def plot_generated_images(generator):
    noise = np.random.normal(0, 1, (16, 100))
    generated_images = generator.predict(noise)
    generated_images = (generated_images + 1) / 2.0

    plt.figure(figsize=(4, 4))
    for i in range(16):
        plt.subplot(4, 4, i+1)
        plt.imshow(generated_images[i, :, :, 0],
cmap='gray')
        plt.axis('off')
    plt.show()

train_gan(generator, discriminator, gan, x_train)
```

8. Observe the Results

- As the training progresses, you'll notice the generated images becoming more and more realistic. Initially, they'll be random noise, but gradually they'll start resembling handwritten digits.

Conclusion

- You've now built your first generative AI model using GANs! This tutorial provided a practical introduction to generative modeling in Python.

Next Steps:

- **Explore Advanced Architectures:** Dive into more complex GANs like DCGANs (Deep Convolutional GANs) and WGANs (Wasserstein GANs).
- **Generate Different Data:** Try generating color images, text, or even music.

Conclusion

Next Steps:

- **Experiment with PyTorch:** Implement the same GAN using another popular deep learning framework.
- **Real-World Applications:** Think about how you can apply generative AI to solve problems in your field.

THANK YOU

- Special thanks to Gemini and ChatGPT for all the help on content
- Follow along for more informative articles in Generative AI space