Great! Let's begin your **15-example GAN program using MLP**, tailored to give you expertise from basics to advanced, especially for **Si

* ✅ Concept
* 💭 Hack
* 📌 Real-world SCA analogy
* 📘 Assignment
* 🧪 Code (with random data)

✅ **Example 1: Basic GAN Using MLP (No SCA Yet)**

🔍 **Goal**:

Understand how a GAN works fundamentally using two simple MLPs (one for generator, one for discriminator).

💭 Concept

* A **GAN (Generative Adversarial Network)** has two parts:

  * **Generator (G)**: Learns to generate fake data similar to real data.
  * **Discriminator (D)**: Learns to distinguish between real and fake data.
* Both are trained in a game-like setup:

  * Generator tries to fool Discriminator.
  * Discriminator tries to not get fooled.

You’ll train on **1D random normal data** and learn how fake data distribution evolves.

🔨 Hack

> Use **LeakyReLU** instead of ReLU to avoid dying neurons in GANs.
> Use **Label Smoothing** (e.g., real=0.9, fake=0.1) to stabilize training.

🌍 Real-World SCA Analogy

> Think of the **Generator** as an attacker generating fake side-channel traces, while the **Discriminator** is a cryptographic device tr

---

💻 Code (GAN on Random Data)

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Data: Real distribution is Normal(0,1)
def real_data(n):
    return np.random.normal(0, 1, (n, 1))

# Generator Model (MLP)
def build_generator():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu', input_dim=10),
        tf.keras.layers.Dense(1)
    ])
    return model

# Discriminator Model (MLP)
def build_discriminator():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu', input_dim=1),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    return model

# Optimizers
opt = tf.keras.optimizers.Adam(0.0002)

# Build models
generator = build_generator()
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=opt)
```

```
# Combined GAN model
z = tf.keras.Input(shape=(10,))
fake = generator(z)
discriminator.trainable = False
validity = discriminator(fake)
gan = tf.keras.Model(z, validity)
gan.compile(loss='binary_crossentropy', optimizer=opt)

# Training Loop
epochs = 1000
batch_size = 32
for epoch in range(epochs):
    # 1. Train Discriminator
    real = real_data(batch_size)
    fake = generator.predict(np.random.normal(0, 1, (batch_size, 10)), verbose=0)

    X = np.concatenate([real, fake])
    y = np.concatenate([np.ones((batch_size, 1)) * 0.9, np.zeros((batch_size, 1))])  # Label smoothing

    d_loss = discriminator.train_on_batch(X, y)

    # 2. Train Generator
    z = np.random.normal(0, 1, (batch_size, 10))
    g_loss = gan.train_on_batch(z, np.ones((batch_size, 1)))  # Generator wants Discriminator to output 1

    if epoch % 100 == 0:
        print(f"Epoch {epoch}, D Loss: {d_loss:.4f}, G Loss: {g_loss:.4f}")

# Plot Generated Data
generated = generator.predict(np.random.normal(0, 1, (1000, 10)), verbose=0)
plt.hist(generated, bins=30, label="Generated", alpha=0.6)
plt.hist(real_data(1000), bins=30, label="Real", alpha=0.6)
plt.legend()
plt.title("Real vs Generated Distribution")
plt.show()
```

📘 Assignment (Day 1)

1. Run the code above and understand:

   * How input random vectors (latent space) are converted into fake data.
   * How the generator and discriminator update their roles over time.
2. Plot:

   * Discriminator loss vs epoch
   * Generator loss vs epoch
3. Modify generator to have:

   * Two hidden layers
   * Use LeakyReLU
   * Observe effect on stability

🔜 What's Next (Example 2 Preview)

We'll start moving towards **SCA-inspired fake trace generation**, using random traces as real samples, and training GAN to replicate the

---

Would you like a ready-to-run Colab notebook link for this code, or are you running it locally?