fase3_3

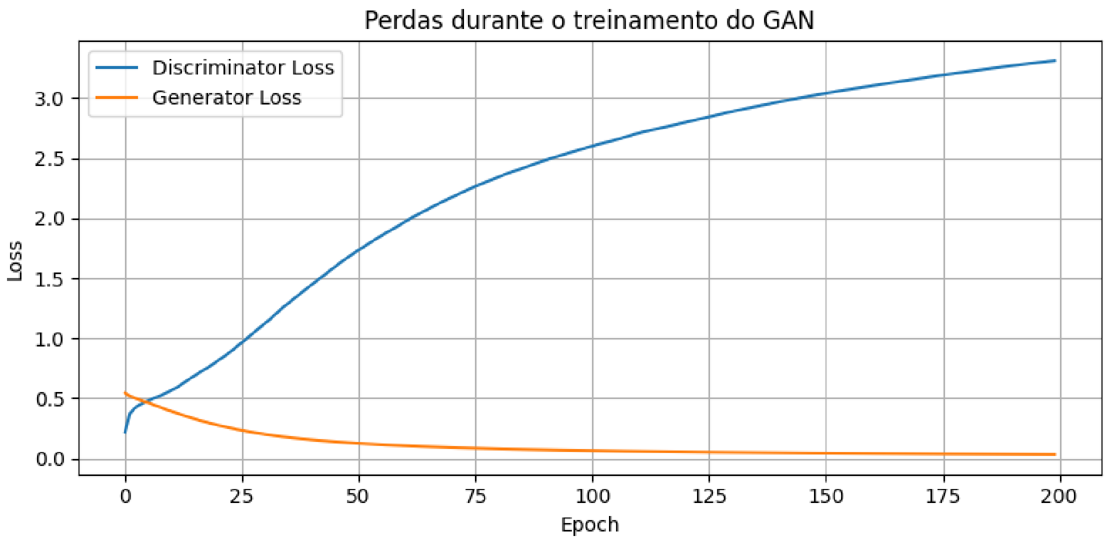
May 18, 2025

[2]:
```
#  GAN corrigido com saída linear no Generator (Solução 1)

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers, Sequential, Input
import matplotlib.pyplot as plt

# ================================
# 1. Dados reais de entrada (X_real)
# ================================
df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/df_simulated.csv")
X_real = df_real.iloc[:, :10].values.astype(np.float32)

n_features = X_real.shape[1]  # 10 WAPs
latent_dim = n_features       # 10

# ================================
# 2. Generator com saída linear
# ================================
def build_generator():
    model = Sequential([
        Input(shape=(latent_dim,)),
        layers.Dense(10, activation='relu'),
        layers.Dense(n_features)  # saída linear
    ])
    return model

# ================================
# 3. Discriminator padrão
# ================================
def build_discriminator():
    model = Sequential([
        Input(shape=(n_features,)),
        layers.Dense(10, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    return model

# ================================
# 4. Compilar modelos
# ================================
generator = build_generator()
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.
 ↪Adam(0.01))

discriminator.trainable = False

gan_input = Input(shape=(latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(0.
 ↪01))

# ================================
# 5. Loop de Treinamento GAN
# ================================
epochs = 200
batch_size = 64
half_batch = batch_size // 2
d_losses, g_losses = [], []

for epoch in range(epochs):
    # Amostras reais
    idx = np.random.randint(0, X_real.shape[0], half_batch)
    real_samples = X_real[idx]
    real_labels = np.ones((half_batch, 1))

    # Amostras falsas
    noise = np.random.uniform(-1, 1, (half_batch, latent_dim))
    fake_samples = generator.predict(noise, verbose=0)
    fake_labels = np.zeros((half_batch, 1))

    # Treinar o discriminador
    d_loss_real = discriminator.train_on_batch(real_samples, real_labels)
    d_loss_fake = discriminator.train_on_batch(fake_samples, fake_labels)
    d_loss = 0.5 * (d_loss_real + d_loss_fake)

    # Treinar o gerador
    noise = np.random.uniform(-1, 1, (batch_size, latent_dim))
    valid_y = np.ones((batch_size, 1))
    g_loss = gan.train_on_batch(noise, valid_y)
```

```python
    # Armazenar perdas
    d_losses.append(d_loss)
    g_losses.append(g_loss)

    if (epoch + 1) % 20 == 0:
        print(f"Epoch {epoch+1}/{epochs} | D_loss: {d_loss:.4f} | G_loss:␣
 ↪{g_loss:.4f}")

# ================================
# 6. Plotar perdas
# ================================
plt.figure(figsize=(8, 4))
plt.plot(d_losses, label="Discriminator Loss")
plt.plot(g_losses, label="Generator Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Perdas durante o treinamento do GAN")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ================================
# 7. Gerar e salvar 40.000 vetores sintéticos
# ================================
n_generated = 40000
noise = np.random.uniform(-1, 1, size=(n_generated, latent_dim))
generated_rssi = generator.predict(noise, verbose=1)

# Clip para faixa realista
generated_rssi = np.clip(generated_rssi, -110, -40)

columns = [f'WAP{str(i+1).zfill(3)}' for i in range(n_features)]
df_generated = pd.DataFrame(generated_rssi, columns=columns)
df_generated.to_csv("/home/darkcover/Documentos/Gan/Data/df_generated.csv",␣
 ↪index=False)
print("  df_generated.csv salvo com sucesso com sinais variáveis")
```

```
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/backend/tensorflow/trainer.py:82:
UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
```

```
Epoch 20/200 | D_loss: 0.7897 | G_loss: 0.2851
Epoch 40/200 | D_loss: 1.4186 | G_loss: 0.1581
Epoch 60/200 | D_loss: 1.9491 | G_loss: 0.1068
Epoch 80/200 | D_loss: 2.3280 | G_loss: 0.0806
Epoch 100/200 | D_loss: 2.5893 | G_loss: 0.0647
Epoch 120/200 | D_loss: 2.7901 | G_loss: 0.0541
Epoch 140/200 | D_loss: 2.9610 | G_loss: 0.0465
Epoch 160/200 | D_loss: 3.0973 | G_loss: 0.0407
Epoch 180/200 | D_loss: 3.2141 | G_loss: 0.0363
Epoch 200/200 | D_loss: 3.3115 | G_loss: 0.0327
```



```
1250/1250              3s 2ms/step
 df_generated.csv salvo com sucesso com sinais variáveis
```