Indian Institute of Technology Hyderabad

Fraud Analytics (CS6890)

Assignment - 4

GAN Generation of Synthetic Data

Name	Roll Number
Saumay Lunawat	CS24MTECH14005
Shivendra Deshpande	CS24MTECH12017
Chinmay Rajesh Ingle	CS23MTECH12002

Contents

List of Figures

1 Problem Statement

The objective of this assignment is to develop a Generative Adversarial Network (GAN) capable of generating synthetic tabular data that accurately mirrors the statistical properties and feature correlations of a given real dataset. The task requires building the Generator and Discriminator architectures using fundamental PyTorch modules, without relying on high-level pre-defined model structures like 'Sequential' applied directly to the entire network (though it's used internally in the provided code for layer sequences). The quality of the generated synthetic data will be evaluated by comparing its feature distributions and pairwise correlations against the original real data.

2 Dataset Description

The real dataset was provided in an Excel file ('data.xlsx'). It consists of tabular data with 10 numerical features. The features are named: 'cov1', 'cov2', 'cov3', 'cov4', 'cov5', 'cov6', 'cov7', 'sal_pur_rat', 'igst_itc_tot_itc_rat', and 'lib_igst_itc_rat'.

The dataset underwent the following preprocessing steps before being used for GAN training:

1. **Loading:** Data was loaded from the Excel file.
2. **Missing Value Imputation:** Any missing values (NaNs) were identified and filled using the median value of their respective columns.
3. **Scaling:** All features were scaled using Scikit-learn's 'StandardScaler', which standardizes features by removing the mean and scaling to unit variance. This scaled data was used for training the GAN.

The number of samples in the original dataset dictates the number of synthetic samples to be generated. The generated synthetic data ('synthetic_data.csv') was inverse-transformed back to the original scale for evaluation purposes.

3 Algorithm

The core algorithm employed is the Generative Adversarial Network (GAN). A GAN consists of two neural networks competing against each other in a zero-sum game framework:

1. **Generator (G):** This network aims to learn the underlying distribution of the real data. It takes a random noise vector (typically sampled from a standard normal or uniform distribution) as input and outputs a data sample that resembles the real data.
2. **Discriminator (D):** This network acts as a binary classifier. It takes a data sample (either real or generated by G) as input and outputs the probability that the sample is real (rather than fake/generated).

Training Objective: The Generator tries to produce data realistic enough to fool the Discriminator, maximizing the probability of the Discriminator misclassifying generated samples as real. The Discriminator tries to improve its ability to distinguish between real samples and fake samples generated by the Generator, maximizing the probability of correctly classifying both real and fake inputs. This adversarial process drives the Generator to produce increasingly realistic data.

The training process typically involves alternating updates to the Discriminator and Generator using a loss function suitable for binary classification, such as Binary Cross-Entropy (BCE) loss, which was used in this implementation.

4 Approach

The implementation followed these steps, as detailed in the 'assignment_4.py' script:

1. **Data Loading and Preprocessing:**
   - Loaded data from 'data.xlsx'.
   - Handled potential missing values using median imputation.
   - Scaled the features using 'StandardScaler'. The scaler object was retained to inverse-transform the synthetic data later.
   - Converted the scaled data into PyTorch Tensors and created a DataLoader for batch processing during training.
2. **Model Architecture:**
   - **Generator:** Implemented as a 'torch.nn.Module'.
     - Input: 100-dimensional noise vector.
     - Architecture: A sequence of Linear layers ('nn.Linear') with hidden dimensions [128, 256, 128]. Each Linear layer (except the last) is followed by Batch Normalization ('nn.BatchNorm1d') and LeakyReLU activation ('nn.LeakyReLU(0.2)').
     - Output: 10-dimensional vector (matching the number of features), with no final activation function (as data is scaled, Tanh was removed).
   - **Discriminator:** Implemented as a 'torch.nn.Module'.
     - Input: 10-dimensional data sample (real or synthetic).
     - Architecture: A sequence of Linear layers with hidden dimensions [128, 64]. Each Linear layer (except the last) is followed by LeakyReLU activation ('nn.LeakyReLU(0.2)') and Dropout ('nn.Dropout(0.3)').
     - Output: A single scalar value passed through a Sigmoid activation ('nn.Sigmoid'), representing the probability of the input being real.
3. **Training Process ('train_gan' function):**
   - Device: Utilized CUDA if available, otherwise CPU.
   - Epochs: Trained for 1000 epochs.
   - Batch Size: 64.
   - Loss Function: Binary Cross-Entropy ('nn.BCELoss').
   - Optimizers: Adam optimizer ('optim.Adam') used for both Generator and Discriminator with learning rates 'lr_g = 0.0002', 'lr_d = 0.0002', and betas '(0.5, 0.999)'.
   - Label Smoothing: Used smoothed labels for the Discriminator training (real label = 0.9, fake label = 0.1) to improve stability.
   - Training Loop:
     - Iterate through epochs and data batches.
     - **Train Discriminator:** - Calculate loss on a batch of real data (target label 0.9). - Generate a batch of fake data using the Generator. - Calculate loss on the detached fake data (target label 0.1). - Sum the real and fake losses, perform backpropagation, and update Discriminator weights.
     - **Train Generator:** - Generate a batch of fake data. - Calculate Discriminator's output for this fake data. - Calculate Generator's loss using the *real* label (0.9) as the target (aiming to fool the Discriminator). - Perform backpropagation and update Generator weights.
     - Store and print losses periodically.
4. **Synthetic Data Generation ('generate_synthetic_data' function):**
   - Used the trained Generator in evaluation mode ('generator.eval()').

- Generated a number of synthetic samples equal to the size of the original dataset.
- Generated data in batches to manage memory.
- Inverse-transformed the generated (scaled) data back to the original data scale using the fitted 'StandardScaler'.

5. **Evaluation ('evaluate_synthetic_data' function):**
   - Compared the distributions of each feature between the original (inverse-transformed) and synthetic (inverse-transformed) datasets using histograms (Figure 2).
   - Computed pairwise Pearson correlation matrices for both datasets (Figure 3).
   - Calculated the absolute difference between the two correlation matrices and visualized it as a heatmap (Figure 4). Calculated the Mean Absolute Difference (MAD) of the off-diagonal correlation differences.
   - Computed and saved basic statistics (mean, std dev) and their absolute differences for each feature into 'statistics_comparison.csv'.
6. **Output Files:** Saved training losses plot, distribution comparison plots, correlation comparison plots, correlation difference plot, statistical comparison CSV, and the synthetic data CSV.

5 Results & Observations

The GAN model was trained, and synthetic data was generated and evaluated against the real data.

Figure 1: GAN Training Losses over 1000 Epochs.

Training Performance: Figure 1 shows the Generator and Discriminator losses over 1000 epochs.

- Both Generator (G) and Discriminator (D) losses appear relatively stable throughout the training, suggesting the training process did not diverge significantly.
- The Generator loss consistently stays lower than the Discriminator loss, which is common in GAN training but doesn't necessarily indicate poor performance.
- There are minor fluctuations in both losses, but no drastic spikes indicating major instability or mode collapse are immediately obvious from this plot alone.

Statistical Comparison (Distributions): Figure 2 compares the density distributions of each feature for the real and synthetic datasets.

- For features like 'cov1', 'cov2', 'cov6', and 'cov7', the synthetic data distribution captures the general shape, peak location, and spread of the real data reasonably well, although the synthetic distributions often appear smoother or slightly shifted.
- Features 'cov3', 'cov4', and 'cov5' show more noticeable discrepancies. The synthetic distributions struggle to replicate the specific shapes and ranges, particularly for 'cov5' which seems highly concentrated around 0 in the real data but has a wider, flatter distribution in the synthetic data.
- The ratio features ('sal_pur_rat', 'igst_itc_tot_itc_rat', 'lib_igst_itc_rat') exhibit significant differences. The real data for these features is highly concentrated near specific values (0 or close to 0), while the synthetic data shows much wider and flatter distributions, failing to capture the sharp peaks. This is also reflected in the 'statistics_comparison.csv' where these features have very large differences in standard deviation (StdDev_Diff values of 0.917, 0.151, 0.978 respectively).
- The mean values (from 'statistics_comparison.csv') are generally closer between real and synthetic data compared to the standard deviations, although features like 'cov3' and 'cov5' show larger mean differences (0.073 and 0.020 respectively).

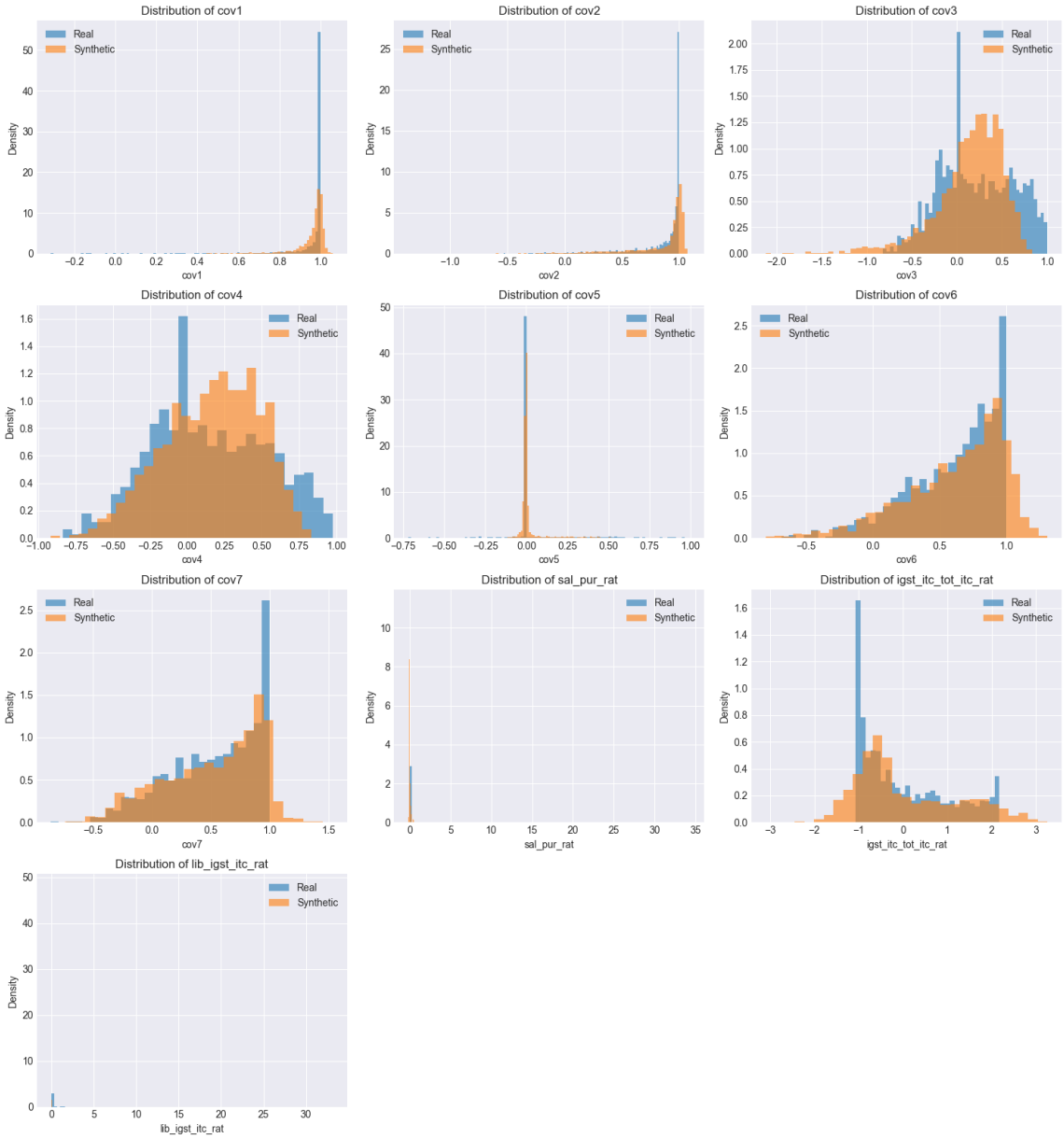

Figure 2: Comparison of Feature Distributions (Real vs. Synthetic Data).

Correlation Comparison: Figures 3 and 4 compare the pairwise feature correlations.

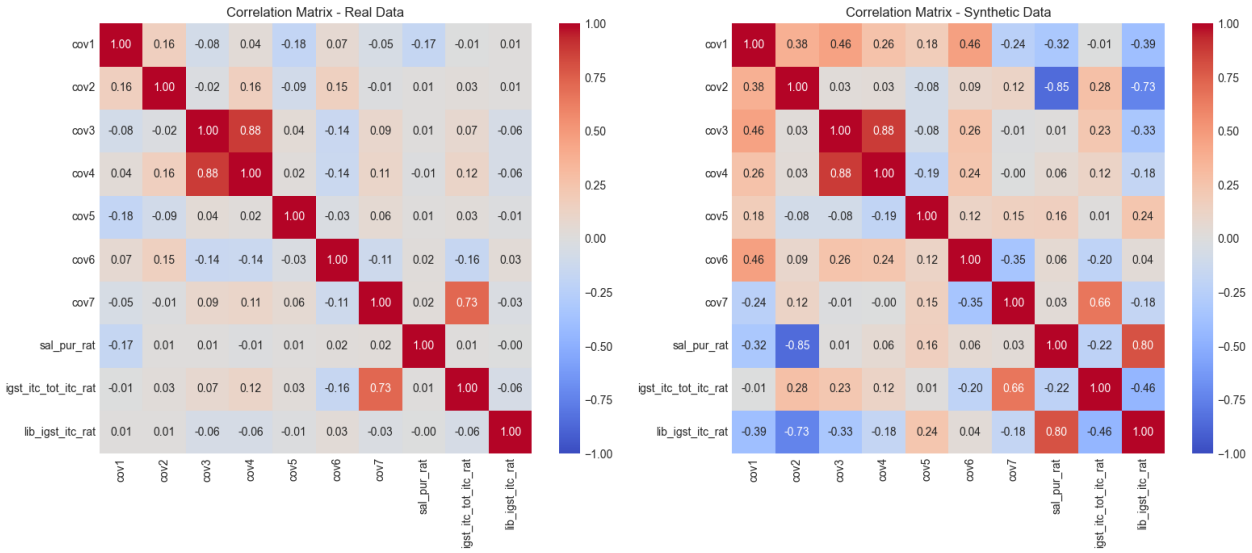

Figure 3: Pairwise Pearson Correlation Matrices for Real and Synthetic Data.

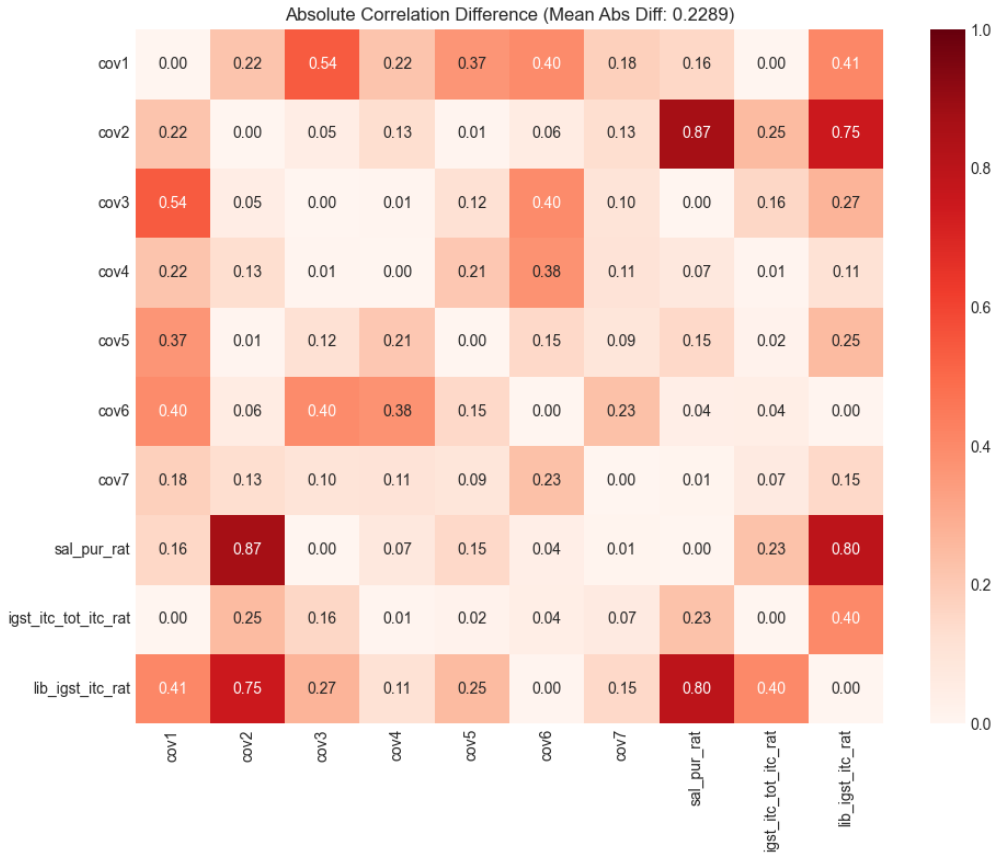

Figure 4: Absolute Difference between Real and Synthetic Correlation Matrices (Mean Abs Diff: 0.2289).

- The overall structure of correlations is somewhat captured, but there are significant differences in the strength and even sign of correlations between the real and synthetic datasets (Figure 3). For instance, the real data shows a strong negative correlation between 'cov7' and 'sal_pur_rat' (-0.73), which is positive (0.03) in the synthetic data. Similarly, 'cov6' and 'cov7' have a moderate positive correlation (0.73) in real data but near zero (0.03) in synthetic data.
- The correlation difference heatmap (Figure 4) highlights pairs with large discrepancies. The largest differences (absolute difference ¿ 0.5) occur for:
  - '(cov2, sal_pur_rat)': Diff 0.87

  - '(lib_igst_itc_rat, cov2)': Diff 0.75
  - '(cov3, cov1)': Diff 0.54
  - '(sal_pur_rat, cov7)': Diff 0.80 (calculation: —(-0.73) - (0.03)— = 0.76, discrepancy with plot value 0.80 - likely rounding in plot vs exact calculation) - let's use the plot's value.

- The Mean Absolute Difference in correlations is 0.2289, indicating a considerable average deviation in pairwise relationships between the real and synthetic data. This suggests the GAN struggled to accurately learn the complex inter-feature dependencies present in the original data.

Overall Observations:

- The GAN successfully generated synthetic data with the correct number of features and samples.
- The model captured the basic marginal distributions (mean, general shape) for some core features ('cov1', 'cov2', 'cov6', 'cov7') reasonably well.
- The model struggled significantly with features having highly peaked or skewed distributions, particularly the ratio features ('sal_pur_rat', 'igst_itc_tot_itc_rat', 'lib_igst_itc_rat'), resulting in much larger standard deviations in the synthetic data.
- The preservation of pairwise correlations was weak, with a relatively high Mean Absolute Difference (0.2289) and several key correlations being poorly replicated or even having the wrong sign.
- The training appeared stable based on the loss curves, but the evaluation metrics indicate that the generator did not fully converge to the true data distribution, especially concerning feature dependencies and the shapes of non-Gaussian distributions.
- Potential areas for improvement could include experimenting with different GAN architectures (e.g., WGAN-GP), adjusting hyperparameters (learning rates, network depth/width), using different noise distributions, or employing more sophisticated feature transformations before training.