Fraud Analytics(CS6890)

Koushik Maji ai23mtech11004
Anirudh Joshi cs23mtech11002
Malsawmsanga Sailo cs23mtech11010

April 23, 2024

Abstract

In this assignment, we aim to explore the potential of Variational Autoencoders (VAEs) in the realm of **synthetic data generation**. Specifically, we focus on the generation of synthetic credit card transaction data. The task involves developing a VAE-based model trained on a given dataset of credit card transactions. The model is expected to generate 2 million synthetic transactions that closely resemble the statistical properties of the original data. The performance of the model will be evaluated based on the distribution of each column in the synthetic data compared to the real data, as well as other metrics discussed in class.

1 Introduction to VAE

Consider a generative model with a conditional distribution $p(x|z, w)$ over the D-dimensional data variable x governed by the output of a deep neural network $g(z, w)$. For example, $g(z, w)$ might represent the mean of a Gaussian conditional distribution. Also, consider a distribution over the M-dimensional latent variable z that is given by a zero-mean unit-variance Gaussian:

$$p(z) = N(z|0, I).$$

To derive the VAE approximation, first recall that, for an arbitrary probability distribution $q(z)$ over a space described by the latent variable z, the following relationship holds:

$$\ln p(x|w) = L(w) + KL(q(z) \parallel p(z|x, w)).$$

where L is the evidence lower bound, or ELBO, also known as the variational lower bound, given by

$$L(w) = \int \frac{p(x|z, w)p(z)}{q(z)} \ln q(z)dz.$$

2 Data Preprocessing

The preprocessing of the data was performed in several steps to ensure that the data is suitable for training the model. The steps are as follows:

1. **Importing the Data:** The data was imported using the pandas library.
2. **Conversion to Numeric Format:** The 'Amount' column, which was initially in a non-numeric format, was converted to a numeric format. This was achieved by replacing the dollar sign and comma characters and converting the result to a float.
3. **Label Encoding:** The categorical columns in the data were label encoded. This was done using the LabelEncoder class from the sklearn library. The 'Amount' and 'Time' column was excluded from this step as it is a numerical column and as time is converted to minute format.
4. **Handling Missing Values:** Any missing values in the data were filled using the mean of the respective column.
5. **Normalization:** The numerical columns in the data were normalized using the StandardScaler class from the sklearn library. The columns that were normalized include 'User', 'Card', 'Year', 'Month', 'Day', 'Time', 'Amount', 'Use Chip', 'Merchant Name', 'Merchant City', 'Merchant State', 'Is Fraud?', 'Zip', 'MCC', and 'Amount'.
6. **Conversion to PyTorch Tensors:** Finally, the preprocessed data was converted to PyTorch tensors for further processing and model training.

The preprocessing steps ensured that the data is in a suitable format and is ready to be used for training the model.

3 Model Definition and Training

The model used in this assignment is a Variational Autoencoder (VAE), which is a type of generative model. The VAE is implemented using PyTorch, a popular deep learning library.

3.1 Model Definition

- **Encoder:** The encoder network takes the input data and maps it to a latent space. It consists of a sequence of linear layers, batch normalization layers, and Tanh activation functions. The output layer of the encoder has twice the dimensionality of the latent space, as it represents the mean and variance of the Gaussian distribution in the latent space. The encoder can be represented as follows:

$$
\begin{aligned}
&\text{Linear}(input\_dim, 512)\\
&\text{BatchNorm1d}(512)\\
&\text{Linear}(512, 256)\\
&\text{BatchNorm1d}(256)\\
&\text{TanhActivation}()\\
&\text{Linear}(256, 128)\\
&\text{BatchNorm1d}(128)\\
&\text{TanhActivation}()\\
&\text{Linear}(128, latent\_dim * 2)
\end{aligned}
$$

- **Decoder:** The decoder network takes points from the latent space and maps them back to the original data space. It also consists of a sequence of linear layers, batch normalization layers, and Tanh activation functions. The output layer of the decoder has the same dimensionality as the input data, as it represents the reconstructed data. The decoder can be represented as follows:

$$
\begin{aligned}
&\text{Linear}(latent\_dim, 512)\\
&\text{BatchNorm1d}(512)\\
&\text{Linear}(512, 256)\\
&\text{BatchNorm1d}(256)\\
&\text{TanhActivation}()\\
&\text{Linear}(256, 128)\\
&\text{BatchNorm1d}(128)\\
&\text{TanhActivation}()\\
&\text{Linear}(128, input\_dim)
\end{aligned}
$$

- **Reparameterization Trick:** The ‘reparameterize‘ function implements the reparameterization trick, which allows the model to backpropagate gradients through the stochastic latent space by sampling an epsilon value from a standard normal distribution and scaling and shifting it by the mean and standard deviation (derived from the log variance) of the current approximate posterior.

- **Forward Pass:** In the forward pass, the model encodes the input data into the latent space parameters (mean and log variance), applies the reparameterization trick to sample from the latent space, and then decodes the sampled latent vector back into the data space. The output is the reconstructed data, along with the mean and log variance of the latent space parameters for use in the loss function (not shown here).

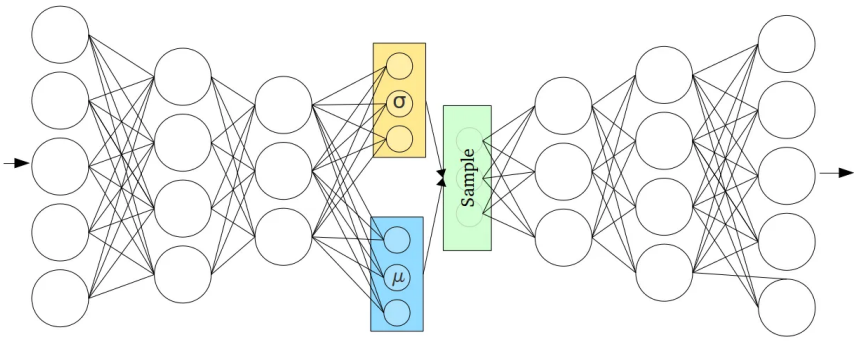

Figure 1: Variational Autoencoders Diagram

3.2 Model Training

The model is trained using the Adam optimizer with a learning rate of 0.001. The loss function used for training is a combination of the reconstruction loss (mean squared error), the Kullback-Leibler divergence, and the DIP loss.

The model is trained for 400 epochs, with a batch size of 400,000. The loss for each epoch is computed and stored in a list for further analysis. The training process involves iterating over the input data in batches.

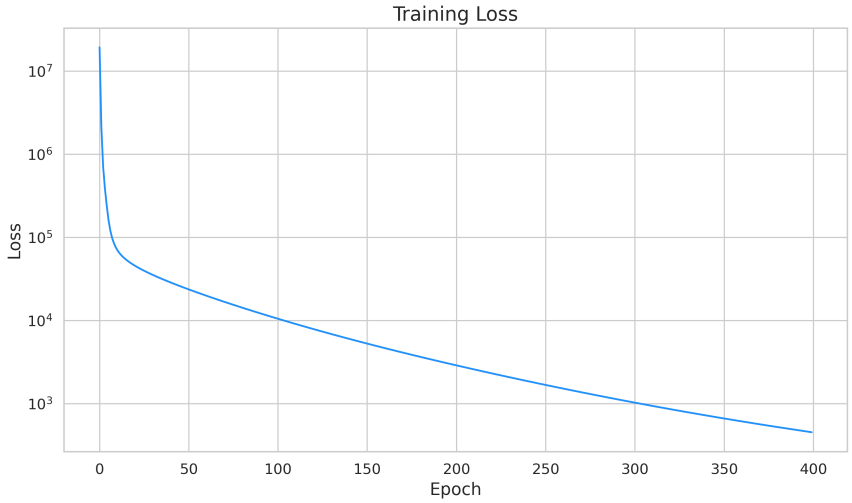

Figure 2: Loss plot for the whole training process

```
# Python code for model training
epochs = 400
batch_size = 400000
for epoch in range(epochs):
    running_loss = 0.0
    for i in range(0, input_data.size(0), batch_size):
```

```
batch_input = input_data[i:i+batch_size].to(device)
optimizer.zero_grad()
recon_batch, mu, log_var = vae(batch_input)
loss = vae_loss(recon_batch, batch_input, mu, log_var)
loss.backward()
optimizer.step()
running_loss += loss.item()
```

4 Data Generation from Latent Space

The method involves the following steps:

1. **Sampling from Latent Space:** We start by sampling a large number of points from a high-dimensional latent space. In this case, we sample 2000000 points from a 128-dimensional latent space. Each point in this space can be thought of as a compact representation of some data. The prior distribution of the data in the latent space is assumed to be a multivariate normal distribution with mean 0 and covariance matrix I, the 128×128 identity matrix, denoted as $N(0, I)$.

2. **Batch Processing:** To efficiently process the data, we divide the sampled points into smaller batches. Each batch contains a fixed number of points, in this case, 1000.

3. **Data Generation:** Each batch of points is then passed through a decoder, which is a neural network that has been trained to generate data from points in the latent space. The output of the decoder is a batch of generated data.

4. **Data Aggregation:** The generated data batches are then aggregated to form a complete set of generated data. This is typically done by concatenating the batches along the first dimension.

5 Evaluation of Synthetic Data Generation

The evaluation of synthetic data generation can be broken down into three levels of granularity:

5.1 Coarse-grained Evaluation

This level of evaluation focuses on the overall similarity between the synthetic data and the real data. Two key metrics are used:

- **Percentage of Direct Copies:** This metric measures what percentage of the synthetic data is a direct copy of the real data. It helps to understand if the model is merely memorizing the training data.

Algorithm 1 DirectCopiesEvaluation

Require: real_data, synthetic_data
Ensure: percentage_direct_copies, percentage_self_copies
1: total_data ← concatenate(real_data, synthetic_data)
2: dup_total ← count_duplicates(total_data)
3: dup_real ← count_duplicates(real_data)
4: dup_synthetic ← count_duplicates(synthetic_data)
5: copies ← dup_total - dup_real - dup_synthetic
6: percentage_direct_copies ← copies / length(total_data)
7: percentage_self_copies ← dup_synthetic / length(synthetic_data)

- **Percentage of Self Copies:** This metric measures what percentage of the synthetic data is a copy of another synthetic data point. It helps to understand the diversity of the generated data.

Algorithm 2 SelfCopiesEvaluation

Require: synthetic_data
Ensure: percentage_self_copies
1: dup_synthetic ← count_duplicates(synthetic_data)
2: percentage_self_copies ← dup_synthetic / length(synthetic_data)

Metric	Percentage
Direct Copies	0%
Self Copies	0%

Table 1: Percentage of Direct and Self Copies

5.2 Medium-grained Evaluation

This level of evaluation involves comparing the distribution of the real data versus the synthetic data. Tools like histograms or density plots can be used to visualize and compare the overall distribution of the data.

- **Distribution Comparison Plots :** In this section, we present a series of plots comparing the distributions of the original and synthetic data for each column in our dataset. These plots provide a visual representation of how closely the synthetic data matches the original data in terms of distribution. The columns in our dataset are as follows: 'User', 'Card', 'Year', 'Month', 'Day', 'Time', 'Use Chip', 'Errors?', 'Merchant Name', 'Merchant City', 'Merchant State', 'Is Fraud?', 'Zip', 'MCC', 'Amount'.

(a) User (b) Card (c) Year

(d) Month (e) Day (f) Time

(g) Use Chip (h) Errors? (i) Merchant Name

(j) Merchant City (k) Merchant State (l) Is Fraud?

(m) Zip (n) MCC (o) Amount

Figure 3: Comparison of data distributions for all columns.