Privacy Preserving Machine Learning - A Study on Private Classification and VAE-Based Data Synthesis

Omri Benbenisty, Yinon Kedem

March 2025

Abstract

In this study, we explore two approaches to integrating differential privacy (DP) into machine learning workflows: training a classifier with differential privacy guarantees and generating a differentially private synthetic dataset to train a classifier. We utilize Variational Autoencoders (VAEs) and the Opacus library to ensure privacy during the training process. The first approach involves directly training a classifier with differential privacy, ensuring that the model parameters do not leak sensitive information. The second approach involves training a VAE with differential privacy to generate a synthetic dataset that mimics the original data distribution, which is then used to train a classifier without privacy constraints. We compare the performance, privacy guarantees, and practical implications of both methods to provide insights into their effectiveness and trade-offs. Code can be found Here or on: `https://github.com/OmriBenbenisty/PrivacyProject`

1 Introduction

We will explore a `Python` project that leverages Variational Autoencoders (VAEs) and Differential Privacy techniques to generate and classify synthetic data, specifically MNIST. The project is implemented using popular libraries such as `PyTorch, Opacus`, and `WandB` for logging and visualization. We will delve into the key components of the code, including the VAE model, the training process with privacy guarantees, and the generation of synthetic datasets. Additionally, we will discuss the challenges encountered during the implementation and how they were addressed to ensure the proper functioning of the model while maintaining privacy. This article aims to provide a comprehensive understanding of how to integrate differential privacy into machine learning workflows, particularly in the context of generative models.

2 Dataset Creation

We start by generating a synthetic datasets using a Variational Autoencoder (VAE) trained with differential privacy guarantees. The VAE model is designed to learn a latent representation of the MNIST dataset, which allows us to generate new, synthetic samples that resemble the original data. Here are the key steps involved in the dataset generation process:

1. **Training the VAE**: The VAE is trained on the MNIST dataset with differential privacy using the Opacus library. This ensures that the model learns useful representations while preserving the privacy of the training data.
2. **Sampling from the Latent Space**: After training, we sample from the latent space of the VAE. The latent space is a lower-dimensional representation of the data learned by the encoder part of the VAE.

3. **Generating Synthetic Data**: The sampled latent vectors are passed through the decoder part of the VAE to generate new synthetic images. These images are similar to the original MNIST digits but are not exact copies, thus maintaining privacy.

To be able to sample from the latent distribution we use the reparameterization trick. The reparameterization trick is a technique used in Variational Autoencoders (VAEs) to allow backpropagation through the stochastic sampling process. This trick enables the model to learn the parameters of the latent space distribution by making the sampling process differentiable.

In a VAE, the encoder outputs the mean (μ) and the logarithm of the variance ($\log \sigma^2$) of the latent space distribution. The reparameterization trick involves sampling from a standard normal distribution and then scaling and shifting this sample using the mean and standard deviation derived from $\log \sigma^2$.

Mathematically, the reparameterization trick can be expressed as:

$$z = \mu + \sigma \cdot \epsilon$$

where:

- μ is the mean of the latent space distribution.
- $\sigma = \exp(0.5 \cdot \log \sigma^2)$ is the standard deviation.
- $\epsilon \sim \mathcal{N}(0, 1)$ is a sample from a standard normal distribution.

This trick allows gradients to flow through the sampling process, enabling the VAE to be trained using standard backpropagation.

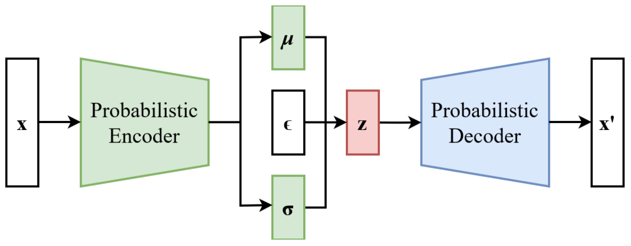

Figure 1: VAE Architecture

After the VAE is trained we use the following code for generating synthetic data:

```
def generate_synthetic_data_with_mean_calculation(vae: VAE, train_loader,
                                                  num_samples_per_class=1000):
    class_means = torch.zeros(10, vae.latent_dim).to(device)
    class_stds = torch.zeros(10, vae.latent_dim).to(device)
    class_counts = torch.zeros(10).to(device)

    vae.eval()
    with torch.no_grad():
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
        _, mu, logvar, _ = vae(images, labels)
        std = torch.exp(0.5 * logvar)
        for i in range(10):
            class_mask = (labels == i)
            class_means[i] += mu[class_mask].sum(dim=0)
            class_stds[i] += std[class_mask].sum(dim=0)
            class_counts[i] += class_mask.sum()

    class_means /= class_counts.unsqueeze(1)
    class_stds /= class_counts.unsqueeze(1)

    # Generate random vectors for each class and decode to generate images
    generated_images = []

    for i in range(10):
        z = class_means[i] + class_stds[i] * torch.randn(num_samples_per_class,
                                                        vae.latent_dim).to(device)
        generated_images.append(vae.decoder(z).view(-1, 1, 28, 28).cpu())

    generated_images = torch.cat(generated_images, dim=0)
    labels = torch.cat([torch.full((num_samples_per_class,), i,
                        dtype=torch.long) for i in range(10)], dim=0)

    return generated_images, labels
```

This function generates `num_samples_per_class` synthetic images for each class `0-9` by first calculating the mean and std for each class based on the training data, then sampling from the latent space and using the decoder to reconstruct the images. The generated images can then be used for various purposes, such as training other models or performing data analysis, while ensuring the privacy of the original dataset.

3 Metrics and Baselines

To evaluate the effectiveness of our approaches, we use several metrics and establish baselines for comparison. The metrics help us understand the performance and privacy guarantees of the models, while the baselines provide reference points for comparison.

3.1 Metrics

1. **Accuracy**: The primary metric for evaluating the performance of the classifier. It measures the proportion of correctly classified samples out of the total samples.

2. **Loss**: The loss function value during training and testing. For the classifier, we use Cross-Entropy Loss, and for the VAE, we use a combination of reconstruction loss and KL divergence.

3. **Privacy Budget (`noise_multiplier`)**: The privacy guarantees provided by the differentially private training process. `noise_multiplier` is a parameter that quantify the privacy loss.

4. **Synthetic Data Quality**: The quality of the synthetic data generated by the VAE, evaluated using visual inspection and the performance of a classifier trained on the synthetic data.

3.2 Baselines

1. **Non-Private Classifier**: A classifier trained on the original MNIST dataset without any privacy guarantees. This serves as the upper bound for accuracy and performance.

2. **Private Classifier**: A classifier trained on the original MNIST dataset with differential privacy guarantees using the Opacus library.

3. **Synthetic Data Classifier**: A classifier trained on the synthetic dataset generated by the differentially private VAE. This helps us understand the trade-offs between privacy and utility.

3.3 Experimental Setup

We conduct experiments to compare the performance of the different approaches. The key steps are as follows:

1. Train a non-private classifier on the original MNIST dataset and record the accuracy and loss.

2. Train a private classifier on the original MNIST dataset using the Opacus library and record the accuracy, loss, and privacy budget.

3. Train a VAE with differential privacy on the MNIST dataset using the Opacus library.

4. Generate a synthetic dataset using the differentially private VAE.

5. Train a classifier on the synthetic dataset and record the accuracy and loss.

6. Compare the results of the different approaches to understand the trade-offs between privacy and utility.

By using these metrics and baselines, we aim to provide a comprehensive evaluation of the effectiveness of integrating differential privacy into machine learning workflows, particularly in the context of generative models and classification tasks.

4 Training

In this section, we describe the training process for both the Variational Autoencoder (VAE) and the classifier. The training process is designed to ensure that the models learn effectively while maintaining differential privacy guarantees.

4.1 Training the VAE

The VAE is trained on the MNIST dataset with differential privacy using the Opacus library. The training process involves the following steps:

1. **Initialize the VAE and Optimizer**: The VAE model and the Adam optimizer are initialized.

2. **Privacy Engine**: If differential privacy is enabled, the Opacus PrivacyEngine is attached to the VAE and the optimizer.

3. **Training Loop**: The VAE is trained for a specified number of epochs. In each epoch, the following steps are performed:

  - Forward pass: The input images are passed through the VAE to obtain the reconstructed images, mean, and log variance.
  - Compute Loss: The VAE loss, which includes the reconstruction loss and the KL divergence, is computed.
  - Backward pass: The gradients are computed and the optimizer updates the model parameters.
  - Logging: The training loss is logged for monitoring.

4. **Evaluation**: After each epoch, the VAE is evaluated on the test dataset to monitor the test loss.

5. **Save Model**: The trained VAE model is saved for later use in generating synthetic data.

4.2 Training the Classifier

The classifier is trained on either the original MNIST dataset or the synthetic dataset generated by the VAE. The training process involves the following steps:

1. **Initialize the Classifier and Optimizer**: The classifier model and the Adam optimizer are initialized.

2. **Privacy Engine**: If differential privacy is enabled, the Opacus PrivacyEngine is attached to the classifier and the optimizer.

3. **Training Loop**: The classifier is trained for a specified number of epochs. In each epoch, the following steps are performed:
   - Forward pass: The input images are passed through the classifier to obtain the output logits.
   - Compute Loss: The cross-entropy loss between the predicted logits and the true labels is computed.
   - Backward pass: The gradients are computed and the optimizer updates the model parameters.
   - Logging: The training loss and accuracy are logged for monitoring.

4. **Evaluation**: After each epoch, the classifier is evaluated on the test dataset to monitor the test loss and accuracy.

5. **Save Model**: The trained classifier model is saved for later use.

5 Results

In this section, we present the results of our experiments, including the performance of the classifiers and the quality of the synthetic data generated by the VAE. We compare the accuracy, loss, and privacy guarantees of the different approaches.

5.1 Classifier Performance

We trained three classifiers: a non-private classifier, a private classifier, and a classifier trained on the synthetic dataset generated by the differentially private VAE. The following table summarizes the accuracy and loss of each classifier:

Classifier	`noise_multiplier`	Test Accuracy	Test Loss
Non-Private Classifier	-	0.99	0.03
Private Classifier	0.03	0.975	0.195
Private Classifier	0.06	0.967	0.25
Private Classifier	0.1	0.95	0.34
Synthetic Data Classifier	0.03	0.8	4.08
Synthetic Data Classifier	0.06	0.795	3.48
Synthetic Data Classifier	0.1	0.76	4.2

Table 1: Classifier Performance: Comparing a Non-Private Classifier, 3 Private Classifiers and 3 Non-Private Classifiers trained on synthetic data from the VAE

5.2 Loss Graphs

The following figures show the training and test loss graphs for the non-private classifier, private classifier, and the classifier trained on the synthetic dataset:

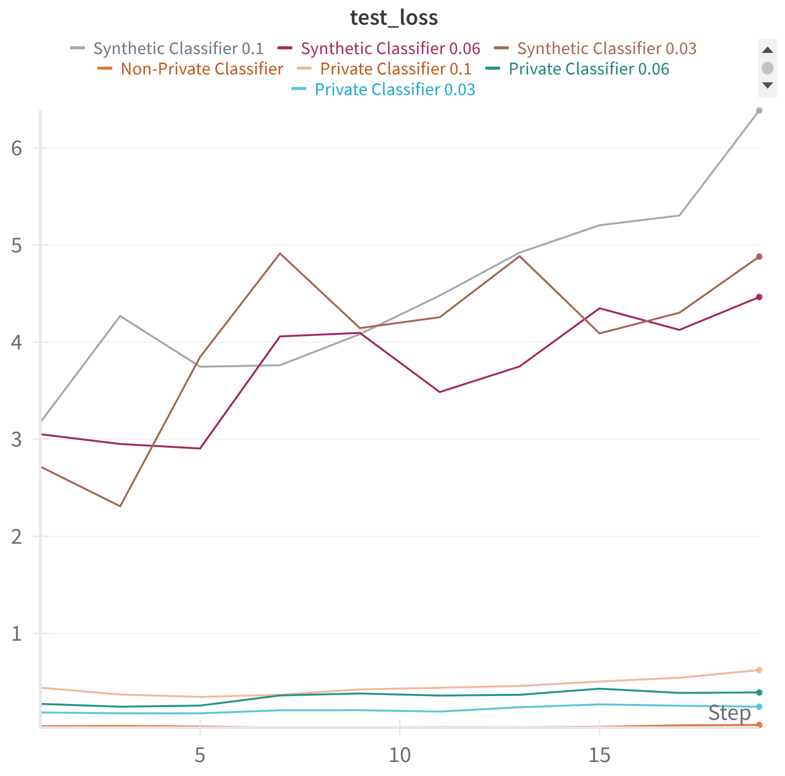

Figure 2: Classifier Test Loss

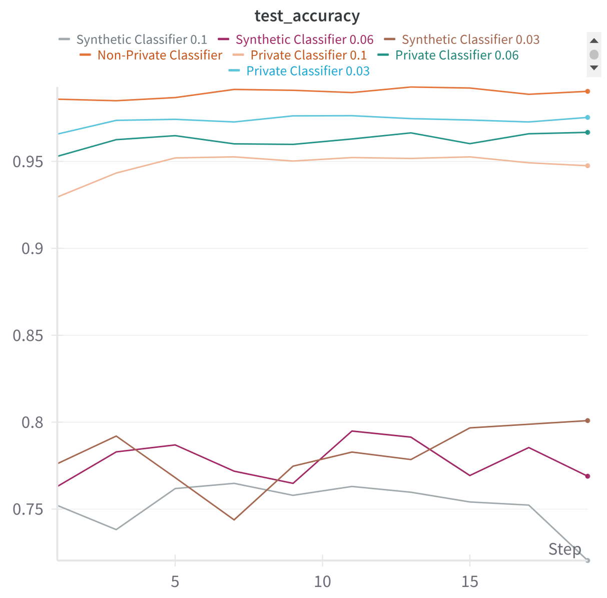

Figure 3: Classifier Test Accuracy

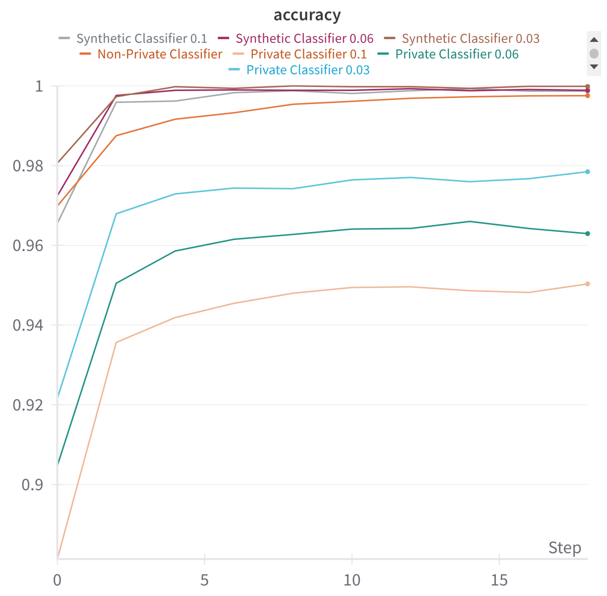

Figure 4: Classifier Train Accuracy

5.3 Synthetic Data Quality

The quality of the synthetic data generated by the VAE is evaluated using visual inspection and the performance of the classifier trained on the synthetic data. The following figure shows example images from the synthetic dataset:

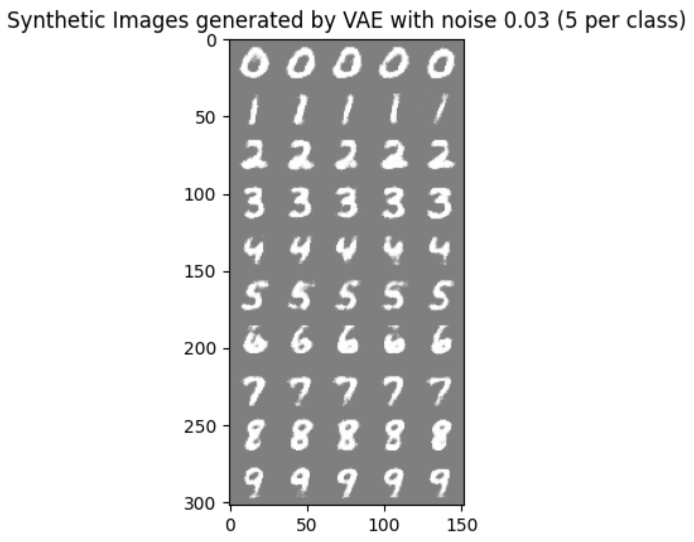
Synthetic Images generated by VAE with noise 0.03 (5 per class)
0
50
100
150
200
250
300
0
50
100
150

Figure 5: Example Images from Synthetic Dataset

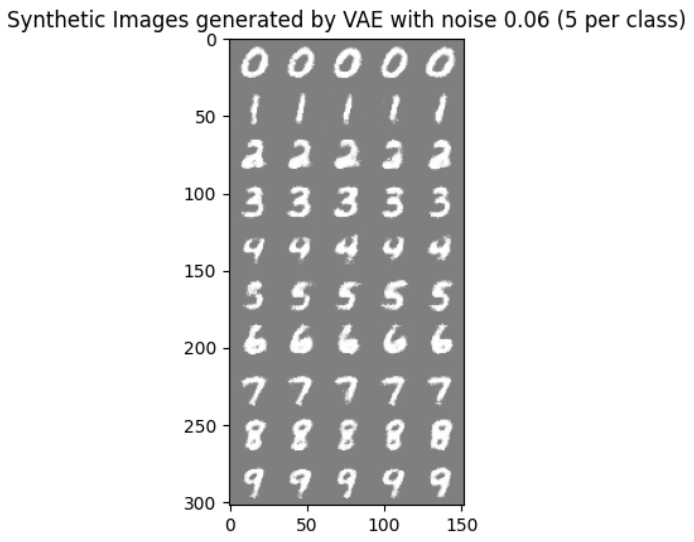
Synthetic Images generated by VAE with noise 0.06 (5 per class)
0
50
100
150
200
250
300
0
50
100
150

Figure 6: Example Images from Synthetic Dataset

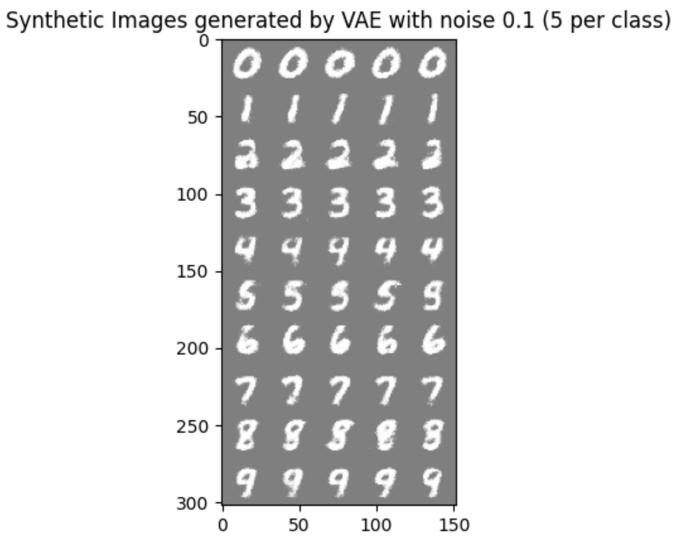
Synthetic Images generated by VAE with noise 0.1 (5 per class)
0
50
100
150
200
250
300
0
50
100
150

Figure 7: Example Images from Synthetic Dataset

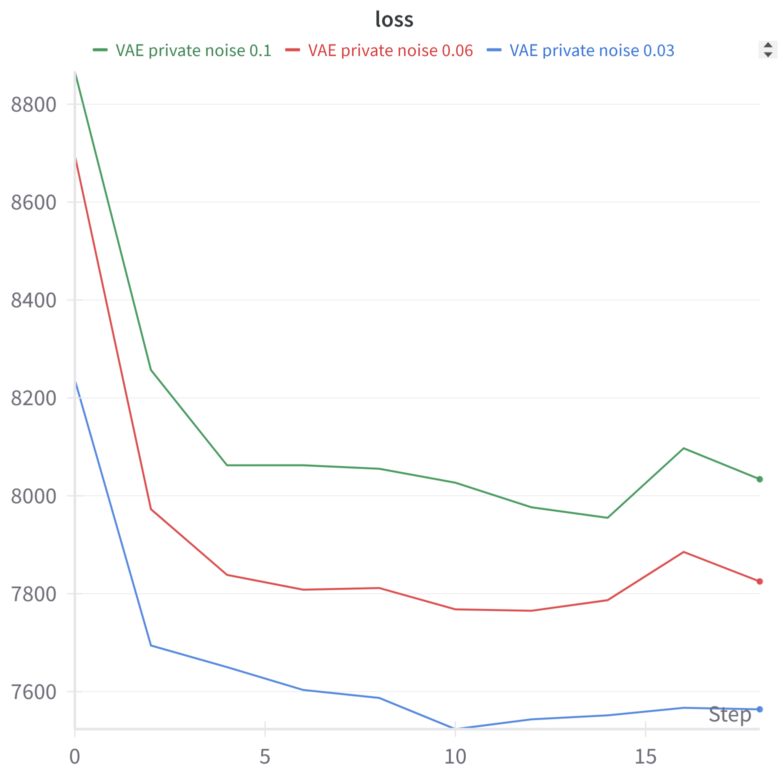

Figure 8: VAE Train Loss

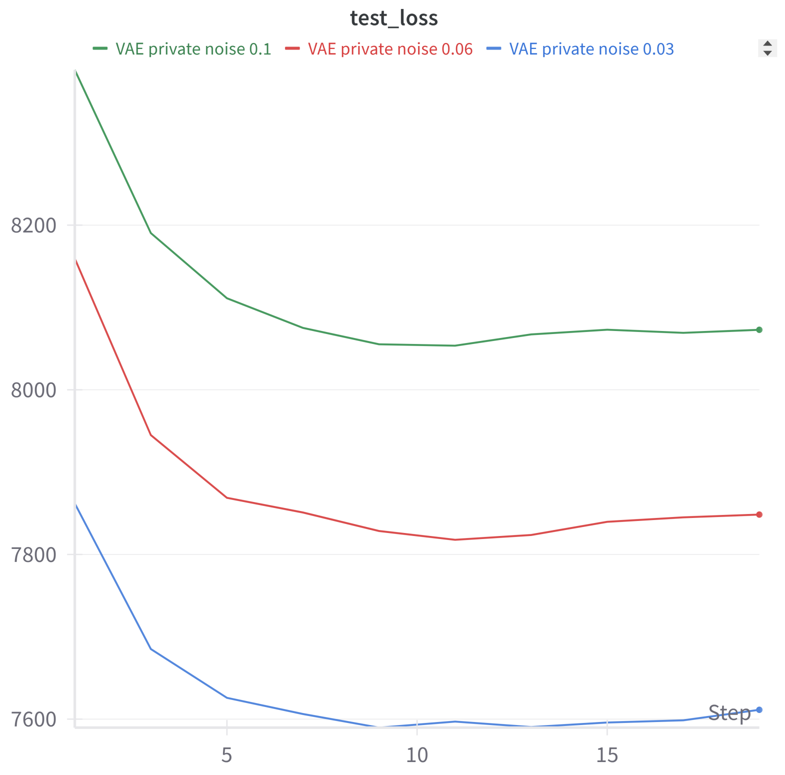

Figure 9: VAE Test Loss

5.4 Discussion

The results show that the non-private classifier achieves the highest accuracy, followed by the private classifier and the classifier trained on the synthetic dataset. The private classifier provides strong privacy guarantees with a reasonable trade-off in accuracy. The synthetic data generated by the VAE is of high quality, as evidenced by the performance of the classifier trained on the synthetic dataset and the visual inspection of the generated images. Even though the classifiers trained on the synthetic datasets performed the worst we believe that with careful tuning, much better results can be achieved as the generated images are very close to the true dataset. Overall, the experiments demonstrate the effectiveness of integrating differential privacy into machine learning workflows, particularly in the context of generative models and classification tasks. The trade-offs between privacy and utility are evident, and the choice of approach depends on the specific requirements of the application.