GAN and VAE as image generator using the mnist dataset

Paulo Cesar S. Maia

Gabriel Dourado

Montreal - 2024

GAN’s

About GAN’s

Generative Adversarial Networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework. This technique was introduced by Ian Goodfellow and his colleagues in 2014 and has since been an active area of research and application.

Here is a breakdown of how GANs operate:

Components: GANs consist of two main parts:

Generator: This network generates new data instances.

Discriminator: This network evaluates them for authenticity; it tries to distinguish between real (training set) and fake (generator’s output) instances.

The ultimate goal of a GAN is to have a generator that perfectly mimics real data, while the discriminator gets better at telling real data apart from the artificially created ones. As training progresses, both networks improve their accuracy and functions, making the discriminator’s job continually challenging.

Training Process:

Generator:

The generator is the part of a GAN that generates new data instances. Its goal is to produce data that is indistinguishable from genuine data.

Input: It begins with a random noise vector (latent space). This randomness allows the generator to explore a wide range of possibilities in the data generation process.

Neural Network: This noise is input into a neural network that transforms it into data with the same dimensions as the training set. For example, if the training data are images of faces, the generator's output would also be images of faces.

Objective: The generator's primary objective is to fool the discriminator into classifying the generated data as real. It continuously improves based on the feedback from the discriminator, learning to produce more accurate and convincing outputs.

Optimization: The generator adjusts its parameters (typically the weights in the neural network) to minimize the difference between its output and the real data, effectively learning the distribution of the training data.

The generator's goal is to fool the discriminator into thinking that the samples it generates are real.

Discriminator:

The discriminator acts as the judge in the GAN setup. It evaluates each instance it receives by classifying it as real (from the actual dataset) or fake (from the generator).

Input: It receives either real data from the training set or fake data generated by the generator.

Neural Network: The discriminator is also a neural network, but with a different architecture and objective compared to the generator. It learns to differentiate real data from fakes by maximizing its ability to correctly classify the received inputs.

Output: Typically, the output is a single scalar value representing the probability that the received data is real. A value close to 1 indicates "real," while a value close to 0 indicates "fake."

Learning: During training, the discriminator gets better at distinguishing real data from the increasingly sophisticated fakes produced by the generator. Its success is essential for providing meaningful feedback to the generator.

Dynamic Interaction

The interaction between the generator and discriminator in a GAN is often described as a competitive game, where both networks improve through the continuous loop of:

- The generator producing better fakes.
- The discriminator improving its ability to detect those fakes.

This training process continues until a point where the generator produces data indistinguishable from real data, and the discriminator is left guessing at random (50/50 chance), unable to improve

further. This state is referred to as Nash Equilibrium, where neither network can unilaterally improve its performance without changing the other's strategy.

Overall, the generator and discriminator in a GAN are designed to push each other towards perfection, providing a powerful framework for learning detailed and complex data distributions in an unsupervised manner.

Challenges in Training GANs

Training GANs can be quite challenging due to several inherent issues:

- **Mode Collapse**: Sometimes the generator starts producing a limited variety of outputs. In extreme cases, it might produce the same output over and over again.
- **Non-Convergence**: The adversarial nature of GANs can lead to oscillation of parameters without reaching a stable solution.
- **Vanishing Gradients**: If the discriminator gets too good compared to the generator, the generator gradients can vanish, stalling training.
- **Hyperparameter Sensitivity**: GANs are notoriously sensitive to the settings of learning rate, batch size, and more, which can make them tricky to configure

Applications:

GANs have been used in various fields such as image generation and editing by creating new images or altering existing ones, e.g., face aging, artistic style transfer. Video generating new video frames to extend existing videos or creating new contextual frames. Voice synthesis: generating human-like speech from text. Data augmentation: generating new data samples to train other machine learning models.

About the dataset:

The Fashion-MNIST dataset, introduced by Zalando Research, is a modern alternative to the traditional MNIST dataset of handwritten digits. Fashion-MNIST consists of 70,000 grayscale images, each 28x28 pixels, divided across 10 fashion categories such as trousers, shirts, coats, and shoes. The dataset is commonly used for benchmarking machine-learning algorithms, especially in image classification tasks.

GAN implementation with fashion mnist:

The code starts by importing necessary libraries such as NumPy for numerical operations, Matplotlib for plotting, and various components from TensorFlow's Keras API for building neural networks. It then loads the Fashion-MNIST dataset, which consists of 28x28 grayscale images of fashion items. The dataset is normalized so that pixel values are scaled between -1 and 1, a common practice for neural network inputs.

GAN Components

- **Generator**: It starts with a dense layer that takes a noise vector of dimension **latent_dim** (100 in this case) as input. This layer is reshaped to form the basis of a convolutional image which is upsampled through transposed convolutional layers until it reaches the desired output size and shape (28x28 pixels). The activation function 'tanh' is used in the output layer, matching the normalization of the input data.
- **Discriminator**: It uses LeakyReLU activation and dropout layers to prevent overfitting. The input to the discriminator is an image (either a real one from the dataset or a fake one from the generator), and the output is a single probability indicating whether the image is real.

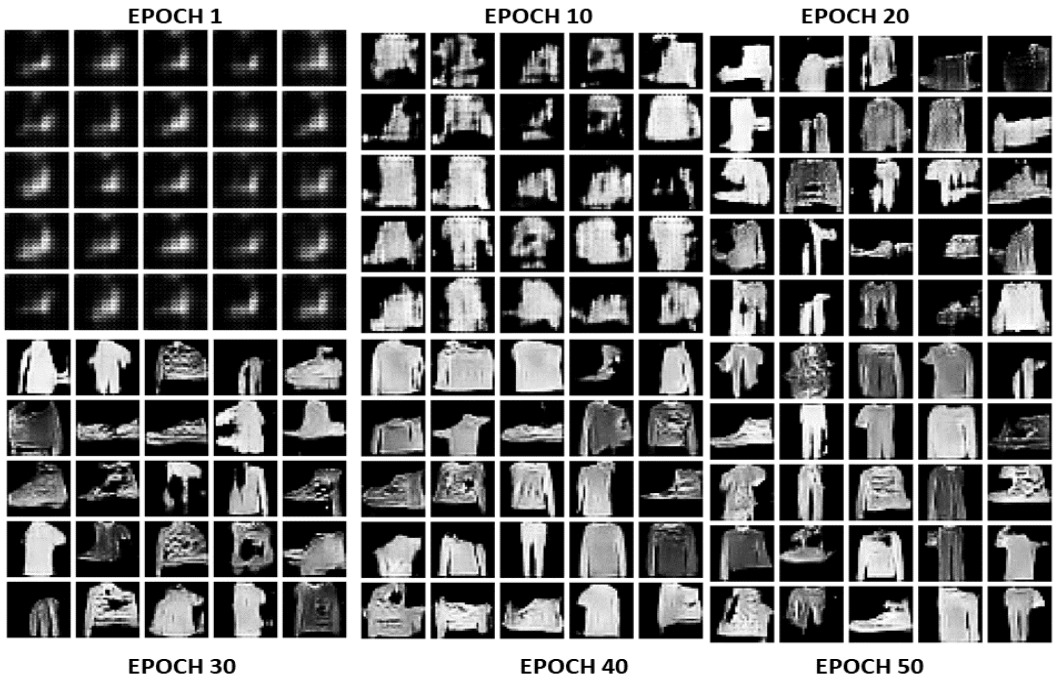
Training Process

The training involves alternating between training the discriminator and the generator:

- **Discriminator Training:** It is trained on both real images from the dataset and fake images produced by the generator. Label smoothing is used for the real images (labels are set to 0.9 instead of 1) to help stabilize training.
- **Generator Training**: The generator's training is framed as a minimax game where it tries to fool the discriminator. The discriminator is set to non-trainable during this phase to ensure only the generator weights are updated. It uses the feedback from the discriminator to adjust its parameters.

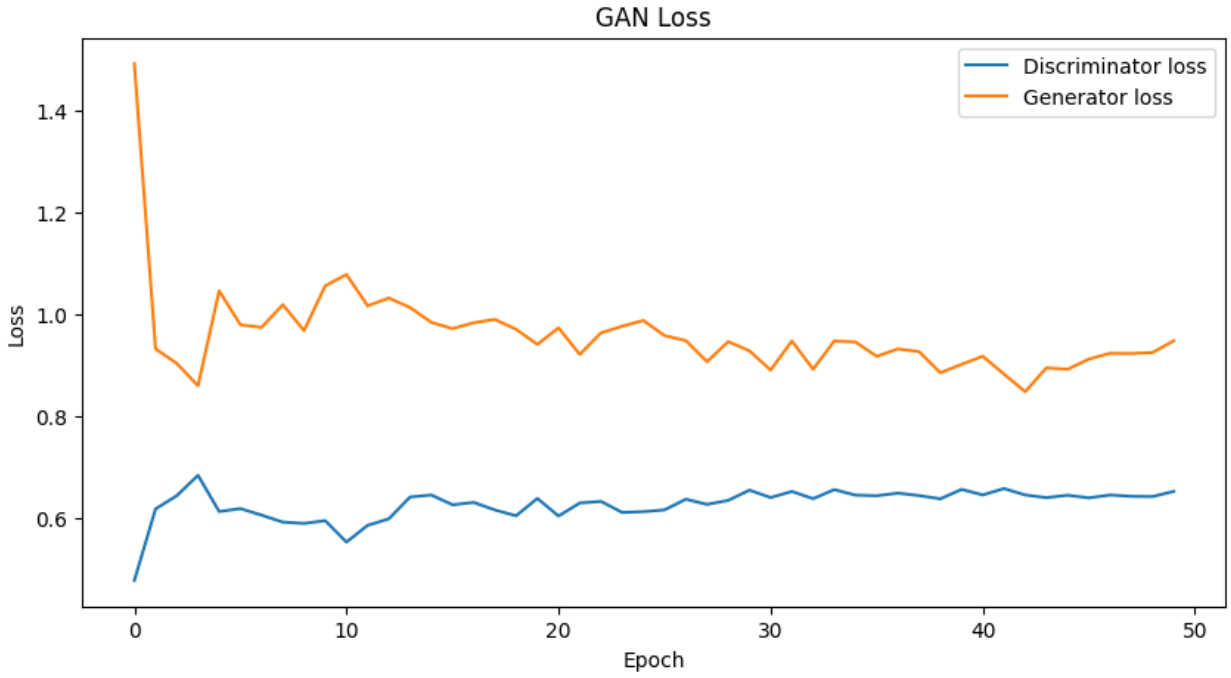
Results:

Here is the results that we got from the GAN with 50 epochs, 245 of batch size and 0.001 of learning rate:



As we can see, the GAN demand more epochs to properly learn and generate good samples.

Here is the loss graph from the discriminator and the generator:



As we can see, the losses starts high, but keep decreasing from each epoch.

To generate good images we need the discriminator loss and the generator in the middle, if one of them get so low is bad for the model. What we can do to get better results is implement a learning rate that are variable, from each epoch. It means that the Lr will be adjusted each time that we get the loss, it is good for the model but is difficult to implement and cost a lot of computational processing.

VAE

1. Overview

The Variational Autoencoder (VAE) model has been trained on the FashionMNIST dataset, aiming to learn a compressed representation of fashion images and to reconstruct these images as accurately as possible. The VAE also attempts to ensure that the latent space (the compressed representation) has good properties that make it useful for generating new data.

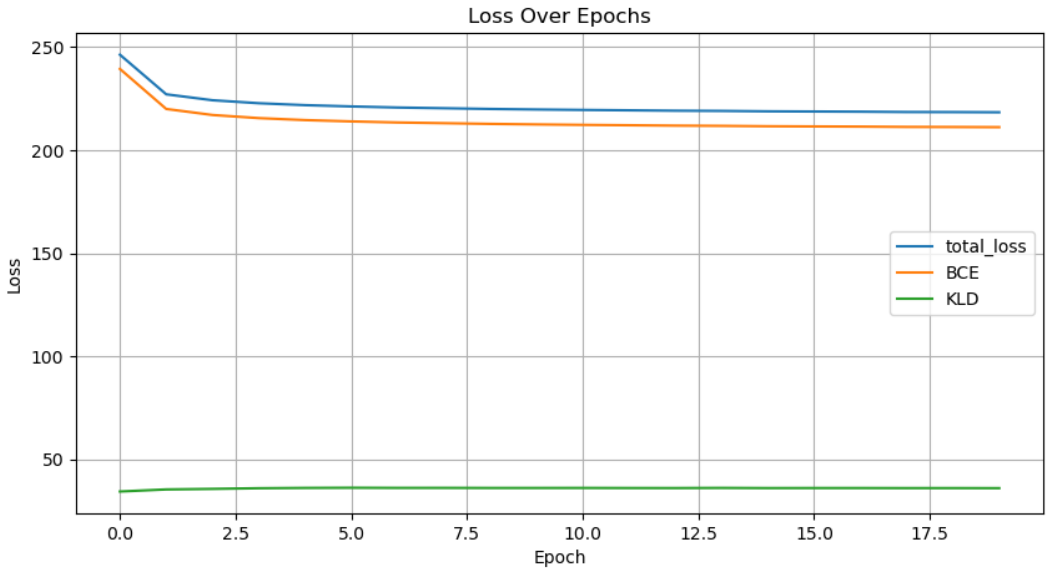
2. Loss Analysis

The training process's effectiveness can be analyzed through the loss components observed across the epochs:

Total Loss: Initially, the total loss decreased sharply, indicating rapid learning and adjustment to the dataset. After the initial epochs, the total loss stabilizes, suggesting that major learning has plateaued, and the model might be approaching its optimal state under the current settings.

Binary Cross-Entropy (BCE): The BCE loss, which measures the difference between the original images and their reconstructions, remains relatively stable, indicating that the model consistently reconstructs images throughout the training process. Consistency in BCE suggests the model's reconstruction capability has matured early in training.

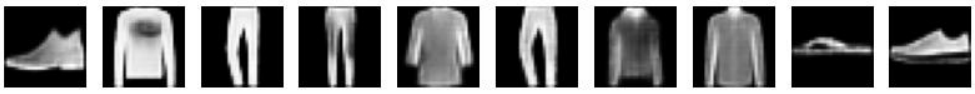
KL Divergence (KLD): The KLD loss is consistently low, indicating that the latent variables' distribution closely matches the prior distribution (assumed to be a standard normal). This is crucial for ensuring that sampled latent variables during generation lead to meaningful outputs.



3. Reconstruction Quality

Original Images (Top Row): These images represent various fashion items like shoes, tops, pants, and bags.

Reconstructed Images (Bottom Row): These are the outputs from the VAE after it has attempted to recreate the original images from the latent representations.



General Observation: The reconstructed images show that the model has learned to approximate the general shape and category of the original items. However, there is a noticeable loss in detail and sharpness across all items.

Specific Observations:

Footwear: The reconstructed shoes maintain their overall shape but lack finer details like laces and distinct sole patterns seen in the originals.

Clothing: Tops and pants show reasonable structural recreation but suffer in replicating textures and subtle design features like pockets, buttons, and logos.

Accessories: Items like bags are less detailed in the reconstructions, missing finer elements like handles and texture.

Common Errors and Patterns

Blurriness: A prevalent issue across all categories is a certain blurriness, which suggests that the decoder might be oversimplifying during the reconstruction phase.

Feature Loss: Specific features like textures and fine details are consistently lost in the reconstruction. This could be due to the limitations in the model's current capacity to capture and recreate complex patterns from the latent space.

4. Latent Space Visualization

Cluster Distribution:

Clarity of Separation: Certain clusters (like the orange and light blue ones) show clear separation, suggesting that the VAE effectively encodes distinguishing features for these classes. This might indicate successful learning of unique characteristics specific to certain types of clothing or items.

Merging of Clusters: Other clusters, particularly those in red and pink, exhibit significant overlap. This indicates potential confusion in the model between these classes. It suggests that the features learned for these classes are not distinct enough to differentiate them effectively in the latent space.

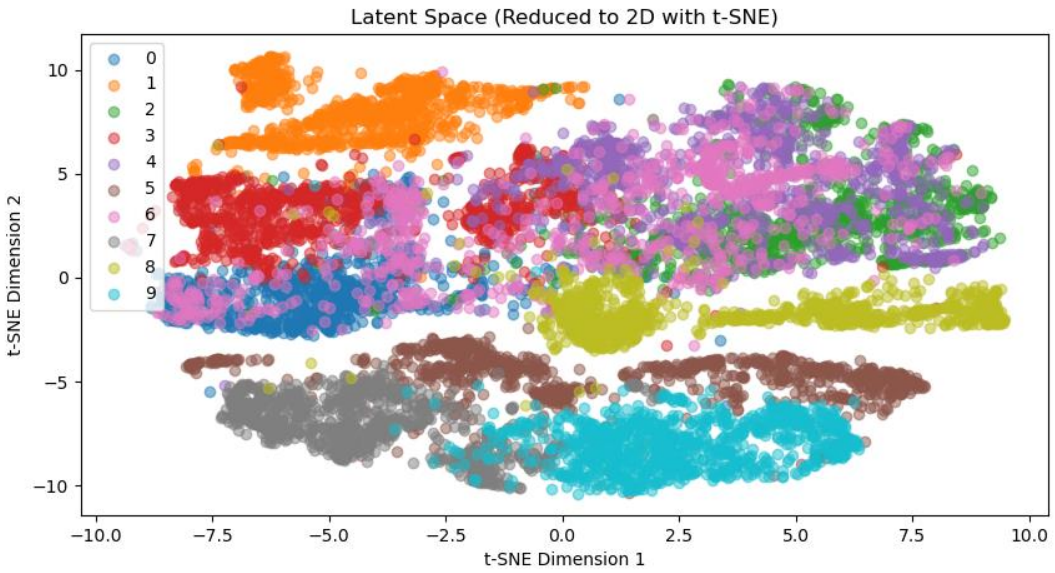
Spatial Organization:

Density and Spread: The spread and density of the clusters provide insight into the variance within each class. Densely packed clusters (such as grey and dark brown) suggest low intra-class variation, meaning the model sees most items within these categories like each other. In contrast, widely spread clusters (like light blue and orange) suggest that the model captures a broader variety of features within these classes.

Outliers and Gaps:

Isolated Points: Outliers or points far removed from their main clusters could represent anomalies in the dataset or items that are not well represented by the model's current architecture. These points can be critical for understanding model limitations or potential data issues.

Gaps Between Clusters: Significant gaps might indicate under-utilized areas of the latent space. These gaps can be opportunities to explore whether additional or different types of regularization might encourage a more efficient use of the latent space.



5. Conclusions and Future Directions

Findings:

Loss Metrics: The early plateau in total loss suggests limitations in the model's learning capacity under the existing setup.

Reconstruction Quality: The lack of fine details and general blurriness in reconstructions indicate that the decoder may need enhancement to improve output fidelity.

Latent Space Analysis: The effective clustering of some classes contrasts with significant overlaps in others, pointing to mixed success in encoding and differentiating data types.

Recommendations:

Enhance Model Architecture: Introducing more complex or deeper layers could help capture finer details.

Adjust Training and Regularization: Optimizing the training regime and exploring advanced regularization techniques could improve model generalization and utilization of the latent space.

Data Augmentation: Implementing more diverse augmentation techniques could expose the model to a wider range of features and variations, potentially improving its learning and generalization capabilities.