Homework3

Variational Auto Encoders

Part 1

Course:

Deep Learning

Supervisor:
Dr. Mansoori

Atefe Rajabi

Spring 2024

Index

Introduction

Blind Source Separation (BSS) is a pivotal technique in signal processing aimed at isolating individual components from a composite signal without prior knowledge of the components or their mixing process. This technique is extensively applied across various fields including audio signal processing and biomedical signal analysis, showcasing its capability to elucidate complex data structures through artificial intelligence methodologies.

Variational Auto Encoder

Variational Autoencoders (VAEs) are a class of generative models that learn to encode data into a compressed latent space and reconstruct it back to the original domain. The model architecture consists of two main components: an encoder that maps the input data to a probabilistic latent space, and a decoder that reconstructs data from this space. By optimizing both the reconstruction loss and the Kullback-Leibler divergence, VAEs ensure a balanced approach to data generation, making them ideal for complex tasks like image separation.

Problem Definition

For Part 1 of this assignment, the challenge lies in employing a Variational Autoencoder to tackle the task of Blind Source Separation with a focus on visual data. Specifically, the task is to separate mixed images, each composed of one MNIST handwritten digit overlaid or blended with one Fashion MNIST clothing item image. The objective is to fine-tune a VAE that accepts these mixed images as input and outputs the distinct images of the handwritten digit and the fashion item, in addition to reconstructing the original mixed image.

Dataset

Data Exploration

The MNIST dataset comprises 60,000 training images and 10,000 testing images of handwritten digits, categorized into 10 classes (0 to 9). Each image is a 28x28 pixel grayscale image. Similarly, the Fashion MNIST dataset includes 60,000 training and 10,000 testing 28x28 grayscale images of fashion items, evenly distributed across 10 different classes (e.g., T-shirt, Trouser, etc.).

Data Visualization

To better understand the characteristics of the data, we visualize random samples from both datasets:

Data Preparation for Mixing

We adopt an image overlay method where each MNIST digit is mixed with a Fashion MNIST item. The images are combined by averaging their pixel values, which ensures the features from both images are preserved but requires normalization to maintain the pixel value range:

mixed_image = (train_images[0] / 2 + fashion_train_images[0] / 2).astype('uint8')

Each mixed image is associated with a tuple of labels representing its original MNIST and Fashion MNIST components.

Data Preprocessing

All images are normalized to ensure their pixel values are scaled to a [0, 1] range, facilitating faster and more stable training. Additionally, adding synthetic noise is considered to simulate more challenging real-world scenarios and improve the model's robustness.

Visualization of Mixed Data

Examples of mixed images are visualized to evaluate the mixing technique's effectiveness and to anticipate potential challenges in the separation task:

Implementation Details:

MixMNIST:

The `MixMNIST` class is designed to generate a novel dataset by combining images from the MNIST and Fashion MNIST datasets. This process involves overlaying corresponding images from each dataset to create composite images, which serve as inputs for training a Variational Autoencoder (VAE) in blind source separation tasks. By transforming the images into tensors and averaging them, the class produces mixed images that challenge the VAE to disentangle the original components. This setup facilitates the development and evaluation of advanced machine learning models for complex signal separation problems, demonstrating the practical application of VAEs in image processing and artificial intelligence.

Encoder Class

The `Encoder` class is responsible for compressing the input data into a latent space representation. It takes high-dimensional input data and passes it through a series of linear transformations, batch normalization, and ReLU activation functions to progressively reduce the dimensionality. The output consists of two sets of means and variances, which represent the latent variables for two different components of the input data. These latent variables capture the essential features needed for reconstruction and separation tasks.

Decoder Class

The `Decoder` class aims to reconstruct the original data from the compressed latent space representation. It takes the latent variables and transforms them back to the original high-dimensional space using a sequence of linear layers, batch normalization, and ReLU activation functions. The final layer uses a Sigmoid activation to ensure the output values are within a specific range, suitable for image data reconstruction.

VAE Class

The `VAE` (Variational Autoencoder) class integrates the encoder and decoder modules to form a complete model capable of both encoding and decoding data. It takes input data, compresses it into latent variables via the encoder, and then generates new samples from these latent variables. These samples are then used by two decoders to reconstruct two separate components of the input data. The VAE class outputs the reconstructed data, individual component reconstructions, and the latent variables' means and variances for further processing and loss calculation.

VAELoss Class

The `VAELoss` class defines the loss function used to train the VAE model. It combines two main components: the reconstruction loss and the Kullback-Leibler divergence. The reconstruction loss measures how well the model can recreate the input data from the latent representation, while the Kullback-Leibler divergence ensures that the latent variables follow a normal distribution, promoting regularization and generalization. The total loss is the sum of these components, guiding the training process to optimize both reconstruction accuracy and latent space regularity.

Function: `train_model`

The `train_model` function is designed to train a Variational Autoencoder (VAE) model using training and validation datasets. It performs several key operations to optimize the model's performance, including forward passes, loss calculation, backpropagation, and model evaluation.

Parameters

- model: The VAE model to be trained.
- train_data: The training dataset containing input data and labels.
- criterion: The loss function used to calculate the difference between predictions and true values.
- optimizer: The optimization algorithm used to update the model's parameters.
- epochs: The number of epochs to train the model.
- val_data: The validation dataset used to evaluate the model's performance.
- device: The device (CPU or GPU) on which the model and data are processed.

1. Model Initialization:
   - The model is moved to the specified device (CPU or GPU).
   - The best model state is initialized, and the minimum validation loss is set to a high value.
   - The model is set to training mode.

2. Training Loop:
   - The function iterates through the specified number of epochs.
   - For each epoch, the training and validation losses are initialized to zero.
   - The model is set to training mode, and the training loop begins.

3. Training Step:
   - For each batch of images and labels in the training dataset:
     - The images and labels are moved to the specified device.

- A forward pass is performed, generating predictions and latent variables.
- The loss is calculated using the criterion.
- The loss value is accumulated.
- Gradients are zeroed, and backpropagation is performed.
- The optimizer updates the model's parameters.

4. Validation Step:

- The model is set to evaluation mode.
- For each batch of images and labels in the validation dataset:
  - The images and labels are moved to the specified device.
  - A forward pass is performed, generating predictions and latent variables.
  - The loss is calculated and accumulated.

5. Epoch Summary:

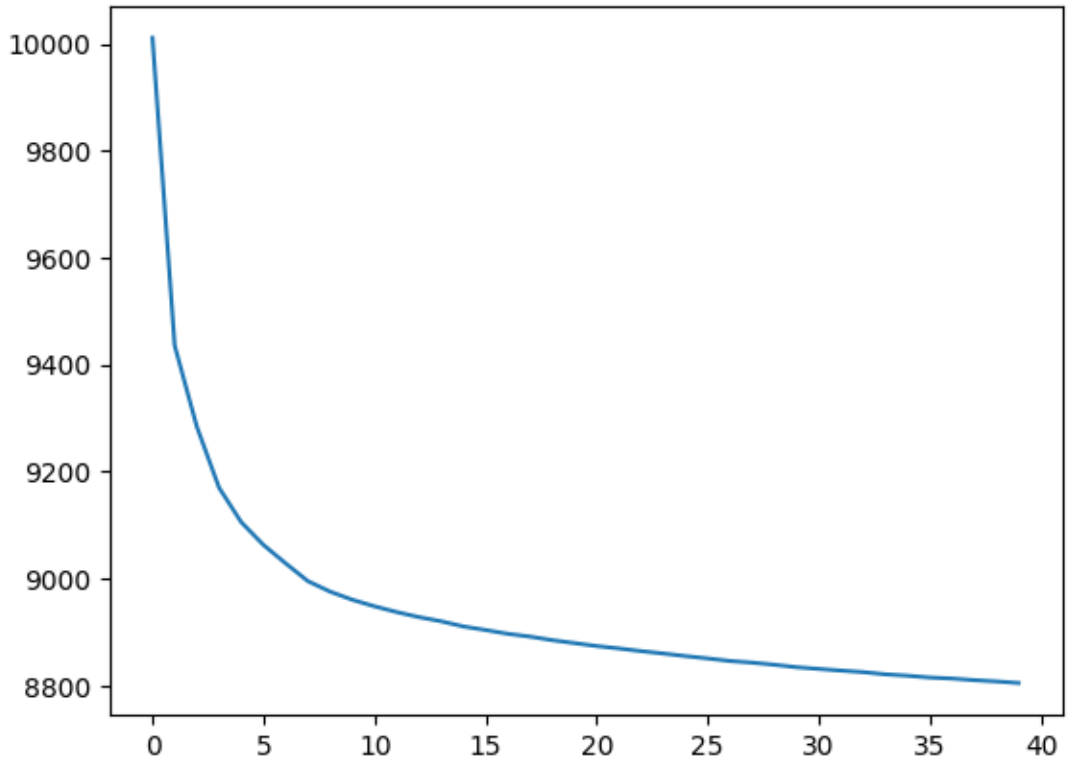
- The average training and validation losses are calculated.
- The losses are appended to their respective lists for later analysis.
- The epoch's losses are printed for monitoring progress.

6. Model Checkpoint:

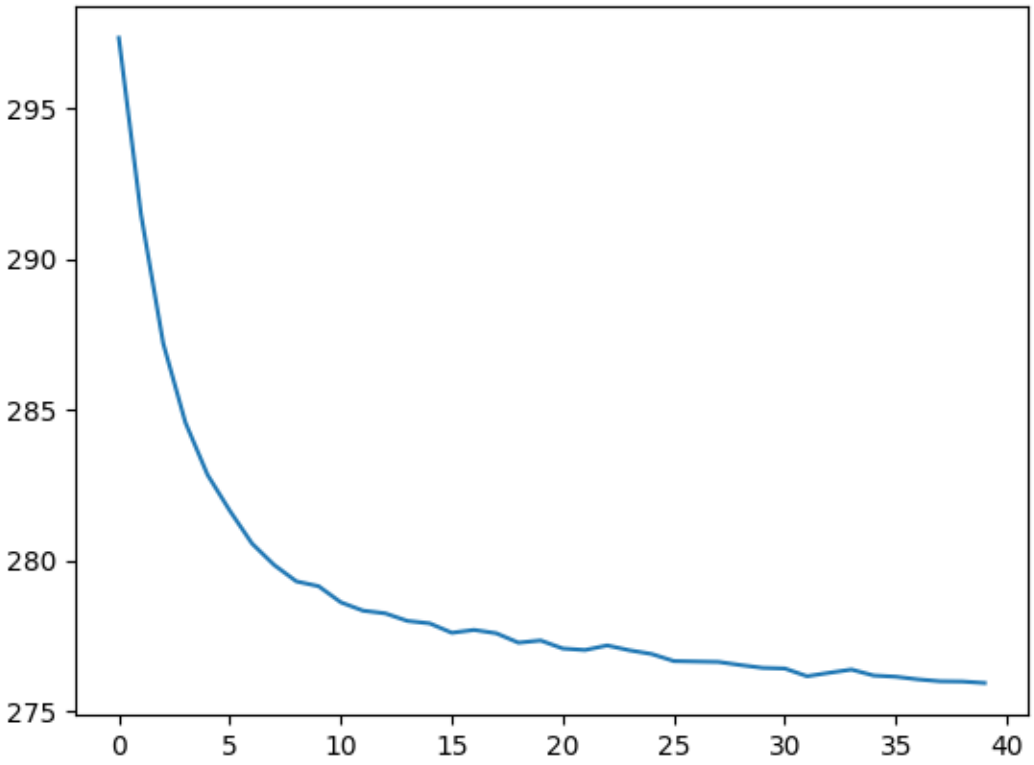
- If the current validation loss is lower than the minimum recorded validation loss, the best model state is updated.
- The model is restored to the best state at the end of training.

Result:

Train Loss:



Validation Loss:



Plot:

First Column: Reconstructed image, Second Column: Fashion image, Third Column: Number image, Forth Column: Main Image