Unsupervised Learning - Final Project

Naama Avni 208523456

Please note the README file for information about the project usage and more.

Question 1

Implementation available on /final_project/encoders/auto_encoder_1_hidden_layer.py

Simple AutoEncoder Implementation Description

This implementation provides a basic autoencoder architecture with a single hidden layer, consisting of a SimpleAutoEncoder class that implements the core neural network with an encoder (input → hidden layer) and decoder (hidden layer → output), and a SimpleAutoEncoderTrainer class that handles the training process.
The architecture uses linear layers with configurable activation functions (ReLU, tanh, or sigmoid) and follows the standard autoencoder pattern where the model learns to reconstruct its input through a compressed latent representation.
The trainer implements standard training procedures including forward/backward passes, loss computation using MSE, and evaluation methods.
The implementation also includes a factory function create_simple_autoencoder() for easy model instantiation and provides methods for both training and inference, including the ability to extract latent representations for downstream analysis. This represents the most basic form of autoencoder architecture, suitable for simple dimensionality reduction and feature learning tasks, with approximately 100K parameters for typical image input dimensions (784 for 28×28 images).

Question 2

Implementation available on /final_project/encoders/auto_encoder_many_hidden_layers.py

Multi-Layer AutoEncoder Implementation

This implementation provides a sophisticated autoencoder architecture with multiple configurable hidden layers, featuring a MultiLayerAutoEncoder class that builds a deep encoder-decoder network with customizable layer dimensions, activation functions, and regularization techniques. The architecture supports dropout for preventing overfitting, batch normalization for training stability, and configurable activation functions including ReLU, tanh, sigmoid, and leaky ReLU.
The MultiLayerAutoEncoderTrainer class handles the training process with additional features like weight decay regularization and methods to extract both latent representations and intermediate encoder features.

This implementation is significantly more complex than the simple version, with approximately 1.1M parameters for typical configurations, making it suitable for more challenging dimensionality reduction tasks and feature learning applications where the simple autoencoder's capacity is insufficient.

Question 3

Implementation available on /final_project/encoders/auto_encoder_many_conv_layers.py

Convolutional AutoEncoder Implementation

This implementation provides a specialized autoencoder architecture designed for image data using convolutional neural networks, featuring a ConvAutoEncoder class that processes 2D spatial data through configurable convolutional layers with pooling operations.
The architecture includes a ConvLayerConfig class for flexible layer specification, supporting various kernel sizes, strides, padding, and pooling configurations. The encoder progressively reduces spatial dimensions while increasing feature channels, followed by a latent layer that flattens the spatial features, and a decoder that reconstructs the original image dimensions using either upsampling or transposed convolutions.
The ConvAutoEncoderTrainer class provides specialized training for image data with methods to extract both latent representations and intermediate convolutional features.
This implementation is optimized for image processing tasks, with approximately 995K parameters, and demonstrates superior performance on image denoising and reconstruction tasks compared to fully-connected architectures due to its ability to capture spatial relationships and local patterns in the data.

Question 4

A. AutoEncoder Denoising Experiment

Experiment Location /final_project/experiments/denoising/

Implementation

This experiment investigates the denoising capabilities and representation learning of the three different autoencoder architectures was implemented on questions 1-3.
The primary goal is to evaluate how well each model can learn to remove noise from input data and to analyze the structure of the latent space they learn.
To provide a controlled and interpretable testbed, the data was synthetically generated to include four distinct pattern types: horizontal lines, vertical lines, diagonals, and random sparse patterns, each represented as 28x28 binary images. Noise of various types (Gaussian, salt & pepper, and speckle) and levels was added to these patterns to simulate real-world corruption.

Each autoencoder was trained to reconstruct the original clean images from their noisy counterparts, allowing for a direct comparison of denoising performance.
The implementation includes comprehensive training, evaluation, and visualization routines: models are trained on the synthetic data, their denoising effectiveness is measured using metrics like MSE, PSNR, and SSIM, and the structure of their learned latent spaces is visualized using t-SNE.
This setup provides insight into how well each model captures and separates the underlying structure of different input patterns.

Results saved to **/final_project/experiments/denoising/denoising_results/**

<u>**Output**</u>

```None
Starting AutoEncoder Denoising Experiment...
============================================================
Generating synthetic data...
Creating autoencoder models...
Creating Simple AutoEncoder...
Creating Multi-layer AutoEncoder...
Creating Convolutional AutoEncoder...
Training models...

Training Simple AutoEncoder (1 Hidden Layer)...
Training Simple AutoEncoder (1 Hidden Layer)...
Epoch 10/50: Train Loss: 0.013761, Val Loss: 0.014543
Epoch 20/50: Train Loss: 0.013224, Val Loss: 0.014364
Epoch 30/50: Train Loss: 0.012989, Val Loss: 0.014297
Epoch 40/50: Train Loss: 0.012874, Val Loss: 0.014292
Epoch 50/50: Train Loss: 0.012777, Val Loss: 0.014264

Training Multi-layer AutoEncoder...
Training Multi-layer AutoEncoder...
Epoch 10/50: Train Loss: 0.024800, Val Loss: 0.020486
Epoch 20/50: Train Loss: 0.019360, Val Loss: 0.016210
Epoch 30/50: Train Loss: 0.018072, Val Loss: 0.016408
Epoch 40/50: Train Loss: 0.017749, Val Loss: 0.016241
Epoch 50/50: Train Loss: 0.017483, Val Loss: 0.016201

Training Convolutional AutoEncoder...
Training Convolutional AutoEncoder...
Epoch 10/50: Train Loss: 0.022288, Val Loss: 0.022255
Epoch 20/50: Train Loss: 0.018813, Val Loss: 0.018176
Epoch 30/50: Train Loss: 0.017806, Val Loss: 0.017398
Epoch 40/50: Train Loss: 0.016813, Val Loss: 0.016802
Epoch 50/50: Train Loss: 0.016210, Val Loss: 0.016198

Evaluating denoising performance...
Evaluating Simple AutoEncoder (1 Hidden Layer)...
  Testing gaussian noise at level 0.1...
  Testing salt_pepper noise at level 0.1...
  Testing speckle noise at level 0.1...
  Testing gaussian noise at level 0.2...
  Testing salt_pepper noise at level 0.2...
  Testing speckle noise at level 0.2...
  Testing gaussian noise at level 0.3...
  Testing salt_pepper noise at level 0.3...
  Testing speckle noise at level 0.3...
  Testing gaussian noise at level 0.4...
  Testing salt_pepper noise at level 0.4...
  Testing speckle noise at level 0.4...
  Testing gaussian noise at level 0.5...
  Testing salt_pepper noise at level 0.5...
  Testing speckle noise at level 0.5...
Evaluating Multi-layer AutoEncoder...
  Testing gaussian noise at level 0.1...
  Testing salt_pepper noise at level 0.1...
  Testing speckle noise at level 0.1...
  Testing gaussian noise at level 0.2...
  Testing salt_pepper noise at level 0.2...
  Testing speckle noise at level 0.2...
  Testing gaussian noise at level 0.3...
  Testing salt_pepper noise at level 0.3...
  Testing speckle noise at level 0.3...
  Testing gaussian noise at level 0.4...
  Testing salt_pepper noise at level 0.4...
  Testing speckle noise at level 0.4...
  Testing gaussian noise at level 0.5...
  Testing salt_pepper noise at level 0.5...
  Testing speckle noise at level 0.5...
Evaluating Convolutional AutoEncoder...
  Testing gaussian noise at level 0.1...
  Testing salt_pepper noise at level 0.1...
  Testing speckle noise at level 0.1...
  Testing gaussian noise at level 0.2...
  Testing salt_pepper noise at level 0.2...
  Testing speckle noise at level 0.2...
  Testing gaussian noise at level 0.3...
  Testing salt_pepper noise at level 0.3...
  Testing speckle noise at level 0.3...
  Testing gaussian noise at level 0.4...
  Testing salt_pepper noise at level 0.4...
  Testing speckle noise at level 0.4...
  Testing gaussian noise at level 0.5...
  Testing salt_pepper noise at level 0.5...
  Testing speckle noise at level 0.5...

Creating visualizations...

Saving results...
Results saved to: denoising_results

============================================================
EXPERIMENT COMPLETED SUCCESSFULLY!
============================================================
================================================================================
=
AUTOENCODER DENOISING EXPERIMENT REPORT
================================================================================
=

EXPERIMENT CONFIGURATION:
----------------------------------------
input_dim: 784
hidden_dims: [512, 256, 128]
latent_dim: 64
conv_input_shape: (1, 28, 28)
batch_size: 64
learning_rate: 0.001
epochs: 50
noise_types: ['gaussian', 'salt_pepper', 'speckle']
noise_levels: [0.1, 0.2, 0.3, 0.4, 0.5]
test_size: 1000

MODEL ARCHITECTURES:
----------------------------------------
Simple AutoEncoder (1 Hidden Layer):
  Parameters: 101,200

Multi-layer AutoEncoder:
  Parameters: 1,153,104

Convolutional AutoEncoder:
  Parameters: 995,009

DENOISING PERFORMANCE SUMMARY:
----------------------------------------

Simple AutoEncoder (1 Hidden Layer):
  Average MSE: 0.034984
  Average PSNR: 15.51 dB
  Average SSIM: 0.5032

Multi-layer AutoEncoder:
  Average MSE: 0.026449
  Average PSNR: 16.13 dB
  Average SSIM: 0.4600

Convolutional AutoEncoder:
  Average MSE: 0.024680
  Average PSNR: 16.35 dB
  Average SSIM: 0.5089

================================================================================
=
```

D. AutoEncoder Anomaly Detection Experiment

Experiment Location /final_project/experiments/anomaly_detection/

Implementation

This experiment evaluates the effectiveness of three autoencoder implemented architectures (simple, multi-layer, and convolutional) for unsupervised anomaly detection on the MNIST dataset.
The task goal is to train models on normal digits (0-8) and detect digit 9 as an anomaly using reconstruction error as the anomaly score.
The implementation follows a standard autoencoder anomaly detection approach: models are trained to reconstruct normal data, then tested on a mixed dataset containing both normal and anomalous samples. The reconstruction error (mean squared error between input and output) serves as the anomaly score, with higher errors indicating potential anomalies.

Results show modest performance across all models, with ROC AUC scores of 0.487 (simple), 0.557 (multi-layer), and 0.502 (convolutional). These low scores are expected and normal for this challenging task, as digit 9 shares many visual features with other digits, making it difficult for simple reconstruction-based methods to distinguish. Interestingly, the convolutional autoencoder achieved the best reconstruction loss (0.010) but bad anomaly detection performance, demonstrating that superior reconstruction quality doesn't necessarily translate to better anomaly detection when the anomaly is semantically similar to normal data.

Final Results and Visualizations saved to /final_project/experiments/anomaly_detection/anomaly_detection_results/

Output

```
None
Training simple autoencoder...
Epoch 5/20, Loss: 0.06514
Epoch 10/20, Loss: 0.05892
Epoch 15/20, Loss: 0.05208
Epoch 20/20, Loss: 0.04521
simple ROC AUC: 0.407

Training multi_layer autoencoder...
Epoch 5/20, Loss: 0.11600
Epoch 10/20, Loss: 0.07211
Epoch 15/20, Loss: 0.05781
Epoch 20/20, Loss: 0.05111
multi_layer ROC AUC: 0.452

Training convolutional autoencoder...
Epoch 5/20, Loss: 0.04694
Epoch 10/20, Loss: 0.02810
Epoch 15/20, Loss: 0.02160
Epoch 20/20, Loss: 0.01880
convolutional ROC AUC: 0.489
```

E. T-SNE Visualization

Experiment Location /final_project/experiments/latent_space/

Implementation

This experiment explores how different autoencoder architectures we created learn to represent the structure of MNIST dataset in low-dimensional latent spaces.
The primary goal is to visualize and compare how well each model can compress and organize the data when restricted to 2D and 3D latent spaces, which is crucial for understanding the quality of learned representations and their suitability for tasks like clustering or visualization. Each autoencoder is trained to reconstruct the original digit images from their compressed latent codes, and the resulting latent vectors are visualized using scatter plots (for 2D and 3D).
The experiment demonstrates how model complexity and latent dimensionality affect the ability to capture meaningful structure in the data: convolutional autoencoders, for example, typically achieve lower reconstruction loss and more informative latent spaces due to their ability to exploit spatial patterns. This setup provides valuable insights into the trade-offs between model simplicity, latent space dimensionality, and the quality of learned representations in unsupervised learning.

All visualisations saved to /final_project/experiments/latent_space/latent_space_results/

<u>Output</u>

```
None
Training simple autoencoder with latent_dim=2...
Epoch 5/50, Loss: 59.84481
Epoch 10/50, Loss: 59.19028
Epoch 15/50, Loss: 58.59775
Epoch 20/50, Loss: 57.71419
Epoch 25/50, Loss: 56.59955
Epoch 30/50, Loss: 51.06471
Epoch 35/50, Loss: 36.98032
Epoch 40/50, Loss: 29.10889
Epoch 45/50, Loss: 24.33769
Epoch 50/50, Loss: 21.17601

Training multi_layer autoencoder with latent_dim=2...
Epoch 5/50, Loss: 57.41585
Epoch 10/50, Loss: 50.80123
Epoch 15/50, Loss: 39.63849
Epoch 20/50, Loss: 27.73590
Epoch 25/50, Loss: 20.20125
Epoch 30/50, Loss: 16.67331
Epoch 35/50, Loss: 15.69838
Epoch 40/50, Loss: 15.60809
Epoch 45/50, Loss: 14.80770
Epoch 50/50, Loss: 14.89141

Training convolutional autoencoder with latent_dim=2...
Epoch 5/50, Loss: 31.90564
Epoch 10/50, Loss: 22.67661
Epoch 15/50, Loss: 19.10170
Epoch 20/50, Loss: 17.60016
Epoch 25/50, Loss: 16.38916
Epoch 30/50, Loss: 15.64890
Epoch 35/50, Loss: 15.49507
Epoch 40/50, Loss: 15.24870
Epoch 45/50, Loss: 15.19315
Epoch 50/50, Loss: 14.70208

Training simple autoencoder with latent_dim=3...
Epoch 5/50, Loss: 53.27124
Epoch 10/50, Loss: 40.89415
Epoch 15/50, Loss: 29.99166
Epoch 20/50, Loss: 22.17574
Epoch 25/50, Loss: 18.50532
Epoch 30/50, Loss: 17.74127
Epoch 35/50, Loss: 17.30742
Epoch 40/50, Loss: 16.92932
Epoch 45/50, Loss: 17.00628
Epoch 50/50, Loss: 16.67022

Training multi_layer autoencoder with latent_dim=3...
Epoch 5/50, Loss: 57.91706
Epoch 10/50, Loss: 51.09178
Epoch 15/50, Loss: 39.83123
Epoch 20/50, Loss: 28.02134
Epoch 25/50, Loss: 19.58762
Epoch 30/50, Loss: 16.40923
Epoch 35/50, Loss: 14.99499
Epoch 40/50, Loss: 14.45038
Epoch 45/50, Loss: 14.20172
Epoch 50/50, Loss: 13.97446

Training convolutional autoencoder with latent_dim=3...
Epoch 5/50, Loss: 31.26838
Epoch 10/50, Loss: 21.01853
Epoch 15/50, Loss: 16.98941
Epoch 20/50, Loss: 15.69422
Epoch 25/50, Loss: 14.87811
Epoch 30/50, Loss: 14.15435
Epoch 35/50, Loss: 13.58436
Epoch 40/50, Loss: 13.09352
Epoch 45/50, Loss: 12.79312
Epoch 50/50, Loss: 12.57226
Results saved to:/final_project/experiments/latent_space/latent_space_results
```