CSE473s: Computational Intelligence – Fall 2025

Project Report

Name	Code
Sara Saber Samuel	2101138
Clara Ashraf Younan	2100932
Bishoy Tarek Soliman	2101067
Mina Ezzat Ragheb	2101363
Mark Matta Guirguis	2100372

Neural Networks From Scratch and Autoencoder-Based Feature Learning

Repository:
https://github.com/MarkMatta03/NeuralNetworkProject

1. Introduction

Neural networks are a core component of modern machine learning systems and are widely applied to tasks such as classification, regression, and representation learning. Although high-level frameworks such as TensorFlow and PyTorch simplify neural network development, they hide many internal details related to learning dynamics, gradient computation, and optimization.

The main objective of this project is to gain a deep understanding of neural networks by implementing them **from scratch using NumPy**, without relying on automatic differentiation.
The project is divided into two main parts:

- **Part 1:** Implementation and verification of a neural network from scratch using the XOR problem.
- **Part 2:** Application of the custom neural network to an autoencoder trained on the MNIST dataset, followed by latent space classification.

The complete implementation, experiments, and source code are publicly available at:
https://github.com/MarkMatta03/NeuralNetworkProject

2. Part 1 — Neural Network From Scratch

2.1 Neural Network Components

A modular neural network library was developed entirely from scratch using NumPy. The library includes:

- Fully connected (Dense) layers
- Activation functions: ReLU, Sigmoid, Tanh, and Softmax
- Mean Squared Error (MSE) loss function
- Stochastic Gradient Descent (SGD) optimizer
- Sequential container for model construction

Each component was implemented independently to ensure flexibility, clarity, and ease of debugging.

2.2 XOR Problem

The XOR problem was used as a benchmark to validate the neural network implementation.
XOR is a non-linearly separable problem, making it a suitable test for verifying non-linear modeling capability.

The network architecture used was:

- Input layer: 2 neurons
- Hidden layer: 4 neurons with non-linear activation
- Output layer: 1 neuron with Sigmoid activation

The model was trained using MSE loss and SGD optimization.

2.3 Training Results

After training for a sufficient number of epochs, the network successfully learned the XOR function.
The final predictions closely matched the expected XOR outputs, and the training loss decreased steadily over time, indicating stable convergence.

2.4 Gradient Checking

To verify the correctness of the backpropagation implementation, gradient checking was performed.
Numerical gradients were computed using the finite difference method and compared with analytical gradients obtained from backpropagation.

The maximum absolute difference between numerical and analytical gradients was on the order of 10^{-5}, confirming that the gradient computation was implemented correctly.

3. Part 2 — Autoencoder on MNIST Dataset

3.1 Dataset Description

The MNIST dataset consists of grayscale images of handwritten digits from 0 to 9. Each image has a resolution of 28×28 pixels and is flattened into a 784-dimensional vector.
Pixel values were normalized to the range [0, 1].

The dataset was split into:

- 60,000 training samples
- 10,000 testing samples

3.2 Autoencoder Architecture

An autoencoder was implemented using the custom neural network library.

Encoder:

- Input layer: 784 neurons
- Hidden layer: 256 neurons with ReLU activation
- Latent layer: 64 neurons with ReLU activation

Decoder:

- Hidden layer: 256 neurons with ReLU activation
- Output layer: 784 neurons with Sigmoid activation

The Sigmoid activation ensures reconstructed pixel values remain within the valid range.

3.3 Training Procedure

The autoencoder was trained using:

- Loss function: Mean Squared Error (MSE)
- Optimizer: Stochastic Gradient Descent (SGD)
- Learning rate: 0.1
- Number of epochs: 20

During training, the model minimized reconstruction error between input images and their reconstructions.

3.4 Reconstruction Results

Original MNIST images were visually compared with reconstructed images.
The reconstructed outputs preserved the overall digit structure and shape, demonstrating that the autoencoder successfully learned meaningful representations of the data.

4. Latent Space Feature Extraction

After training, the encoder portion of the autoencoder was extracted to generate latent feature vectors.
Each image was mapped to a 64-dimensional latent representation, providing a compact and informative description of the original data.

5. Classification Using Latent Features

5.1 SVM Classification

A Support Vector Machine (SVM) with an RBF kernel was trained using the latent representations generated by the encoder.
The classifier achieved high accuracy on the test dataset, demonstrating that the latent features are effective for classification.

5.2 Baseline Comparison

For comparison, an SVM was also trained directly on raw pixel values.
The classifier trained on latent features consistently outperformed the raw-pixel baseline, highlighting the advantage of unsupervised feature learning.

5.3 Evaluation Metrics

Classification performance was evaluated using:

- Accuracy
- Confusion matrix
- Precision, recall, and F1-score

The confusion matrix showed strong diagonal dominance, indicating correct classification for most digit classes.

6. TensorFlow / Keras Comparison

To validate the custom implementation, a reference autoencoder with the same architecture was implemented using TensorFlow/Keras.
Latent features extracted from the TensorFlow model were classified using an SVM.

The classification performance was comparable to the from-scratch implementation, confirming the correctness and effectiveness of the custom neural network library.

7. Conclusion

This project successfully demonstrated the implementation of neural networks from scratch and their application to real-world data.

Key outcomes include:

- Correct implementation of forward propagation and backpropagation
- Verification of gradients through numerical gradient checking
- Effective unsupervised feature learning using an autoencoder
- Improved classification performance using latent representations
- Validation through comparison with TensorFlow/Keras

Overall, implementing neural networks from scratch provided deep insight into learning dynamics and optimization, while framework-based implementations demonstrated efficiency and practicality.

8. Repository Link

The complete source code, experiments, and documentation for this project are available at:
https://github.com/MarkMatta03/NeuralNetworkProject

9. References

- MNIST Dataset
- Scikit-learn Documentation
- TensorFlow/Keras Documentation