FINAL PROJECT

Computational Intelligence Project

Name	ID
Amr Khaled Abd Elsadik	2100524
Arwa Ramadan Abdelfattah	2100647
Hla Ehab Fawzy	2100708
Mario Sameh Samir	2100364
Youssef Ahmed Abd-El-Fattah	2101059

Contents

Introduction

This project focuses on building a complete neural network library from scratch using Python and NumPy, with the goal of understanding and implementing the core mechanisms of deep learning without relying on external frameworks. The library includes modular components for layers, activation functions, loss functions, and optimization, all integrated through a simple network structure designed for easy testing and extension.

library design and architecture choices

The library is structured in a modular and extensible manner, enabling individual components to be easily modified, tested, or expanded. Core modules include:

- **layers.py**: Implements a base Layer class and a Dense layer with weight, bias, and gradient management.
- **activations.py**: Provides activation layers such as ReLU, Sigmoid, Tanh, and Softmax, each implemented as differentiable layers.
- **losses.py**: Contains the Mean Squared Error (MSE) loss, including both the forward computation and the initial gradient w.r.t. predictions.
- **optimizer.py**: Implements a simple yet effective Stochastic Gradient Descent optimizer for parameter updates.
- **network.py**: Defines a Sequential-style class that orchestrates the full forward and backward passes through the network.

This modularity ensures clarity, flexibility, and easy debugging, while closely mirroring the design philosophy of professional deep learning libraries. Each layer encapsulates its own forward and backward logic, and the Network class integrates them into a cohesive training pipeline.

Results from the XOR test

After training, the network successfully classified all four XOR input combinations with high accuracy. The results confirm that the library correctly performs forward propagation, computes gradients, and updates parameters using backpropagation. This initial validation provides a strong foundation for the more complex autoencoder and classification tasks that follow in subsequent sections.

```
predictions: [[-0.96409383]
 [ 0.96353676]
 [ 0.96015632]
 [-0.96415383]]
```

Figure 1: the result of XOR test

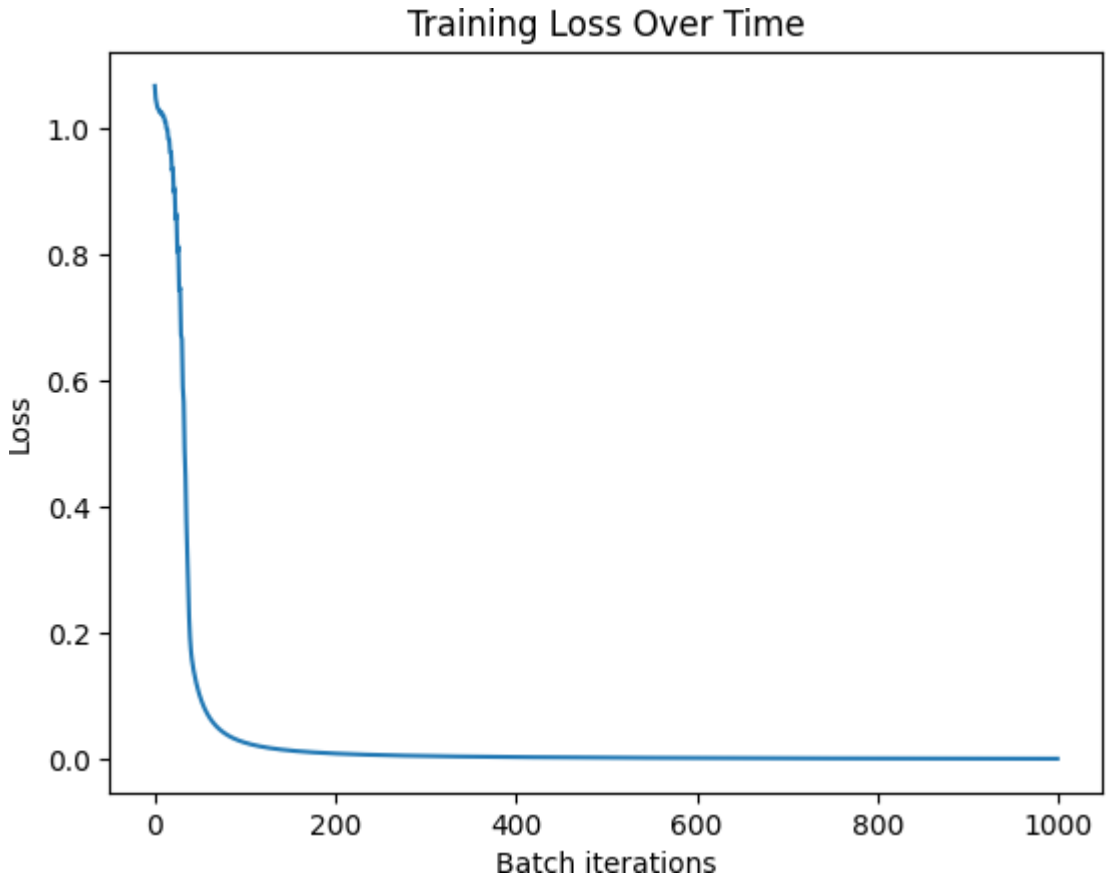


Figure 2: Loss vs iterations

Autoencoder

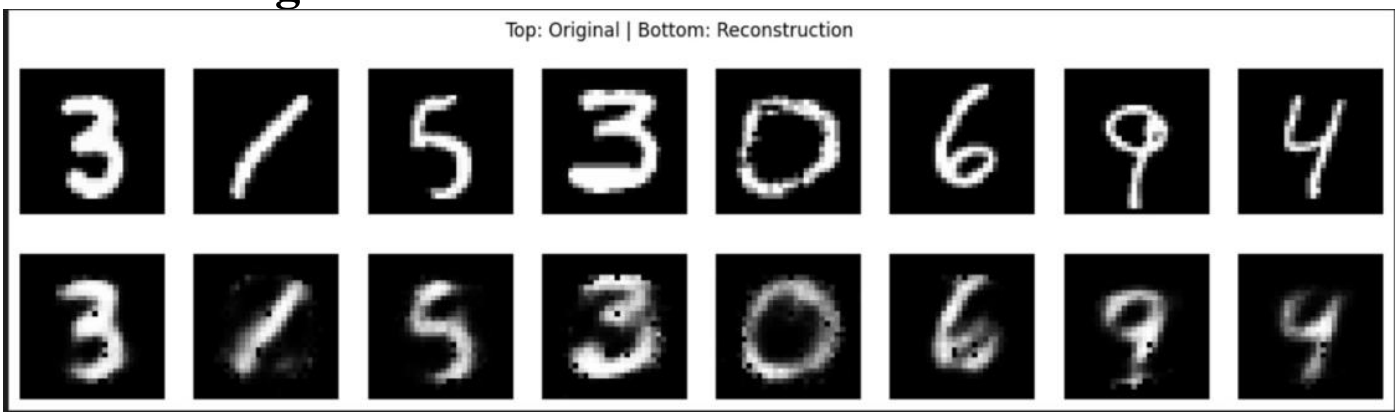
Architecture

- 3-Dense Layers compressing image from 784 pixels to 32 pixels to get important features
- Using RELU activation functions for encoder and decoder layers
- Using Sigmoid for the final activation function maps pixels to be in range of [0,1]
- MSE as Loss Function and SGD as optimizer

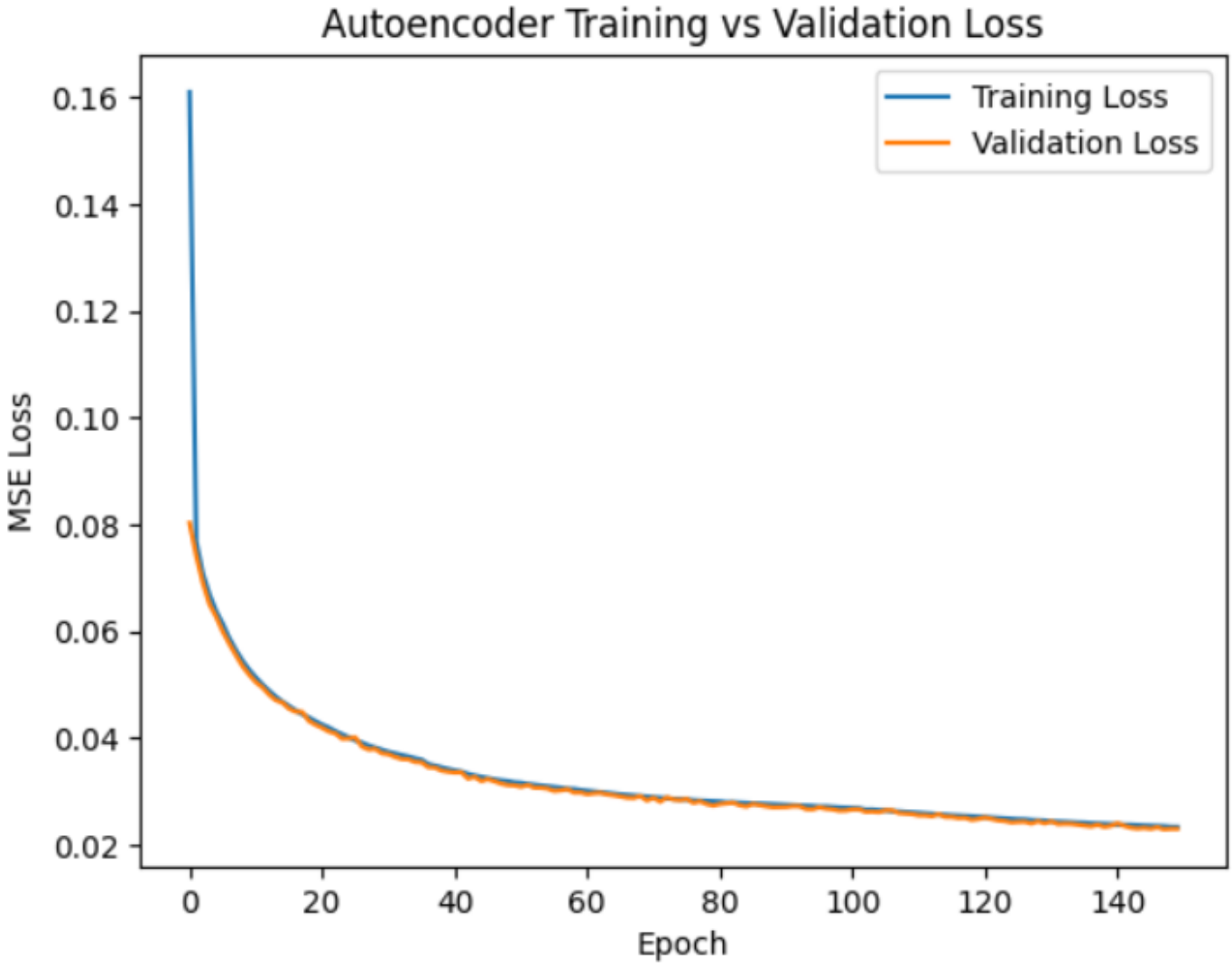
Training

- Mnist data set is divided into training data and validation data
- Data is divided into batches of size 256 each
- Training is done for 150 epochs

Training Results



Loss Curve (training vs validation)



0.1

SVM Classification

```
SVM Accuracy: 0.9403

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.99      0.99      0.99      1135
           2       0.92      0.94      0.93      1032
           3       0.93      0.93      0.93      1010
           4       0.92      0.94      0.93       982
           5       0.91      0.92      0.92       892
           6       0.95      0.96      0.96       958
           7       0.94      0.92      0.93      1028
           8       0.94      0.91      0.92       974
           9       0.92      0.90      0.91      1009

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000


Confusion Matrix:
[[ 963    0    4    1    0    4    5    1    2    0]
 [   0 1122    3    3    0    1    4    0    2    0]
 [   8    0  975    8    6    5    8   13    8    1]
 [   1    0   19  935    0   21    1   13   17    3]
 [   0    0    8    1  924    2   12    5    2   28]
 [   9    0    7   23    5  822   11    2    8    5]
 [  11    3    4    0    3   12  921    0    4    0]
 [   0    6   27    5    5    1    0  947    6   31]
 [   6    1    9   21   15   20    4    7  882    9]
 [   4    7    6    8   41   11    1   15    4  912]]
```

Detailed Analysis

- SVM Training accuracy is acceptable
- Good Average precision and recall results
- Most numbers had good classification, but the model had some difficulties in classifying numbers like (5,2,9)

TensorFlow vs Our library

XOR Problem

1- Ease of Implementation

- TensorFlow was easier due to its ready-to-use built-in functions

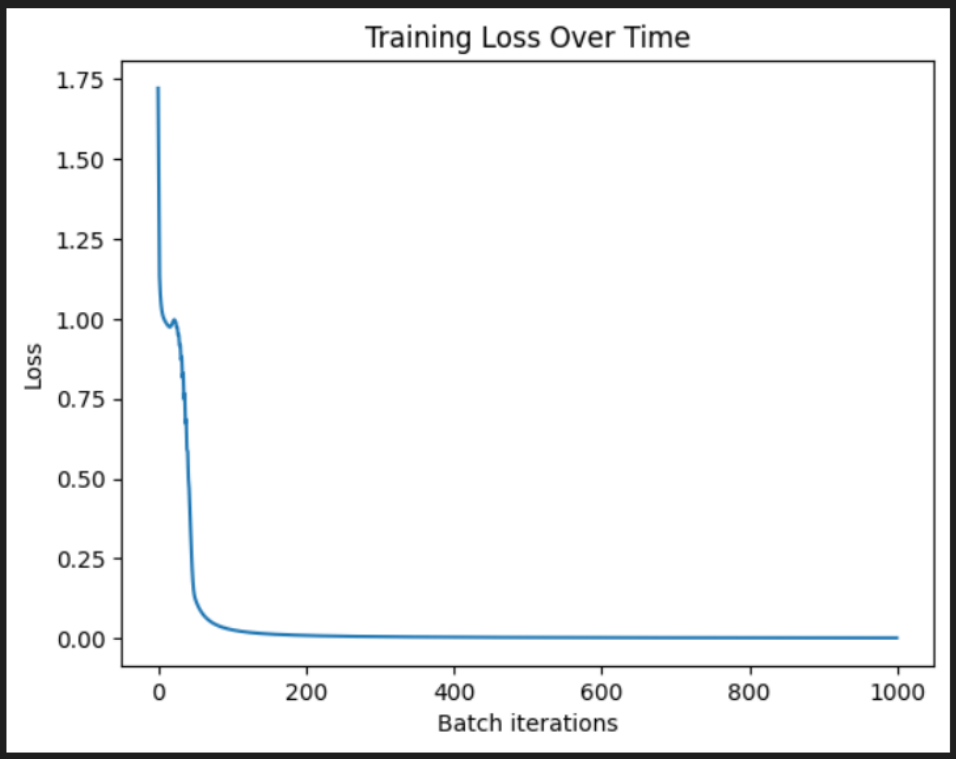
P.O.C	Library	TensorFlow
Training Time (secs)	2	30
Loss	0.001121	0.002969

2- Results

a) Library

```
predictions: [[-0.97408448]
 [ 0.96652033]
 [ 0.96426023]
 [-0.96080841]]
```

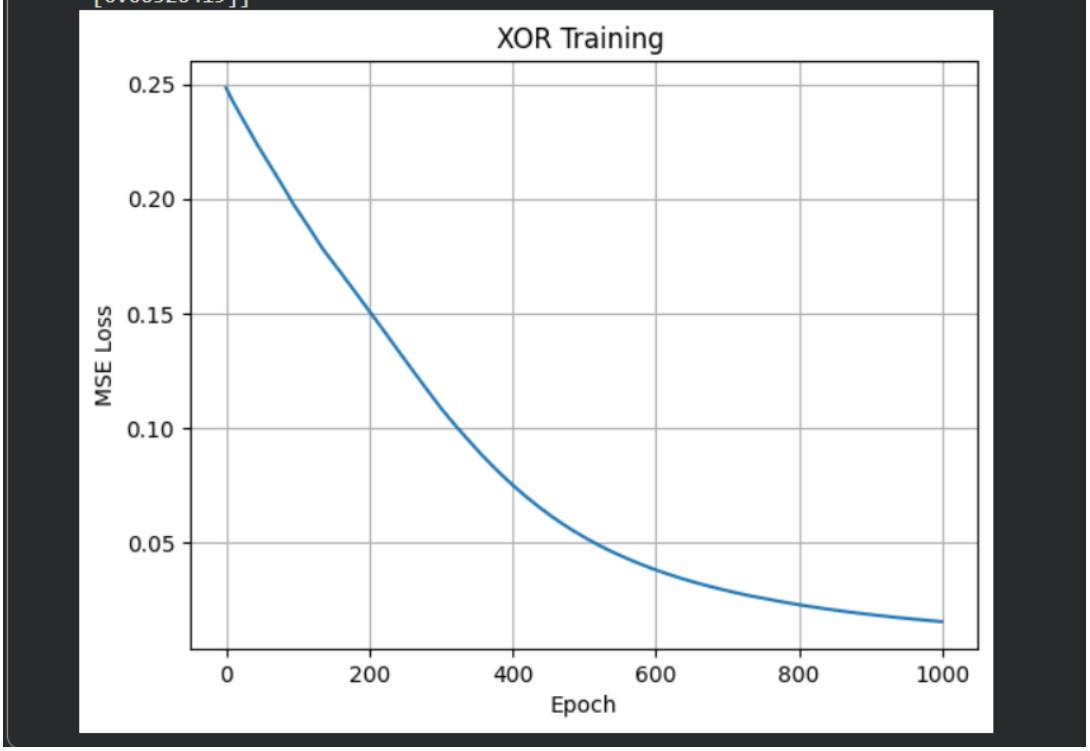
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



0.1

B) TensorFlow

```
Predictions:
[[0.19962347]
 [0.9040187 ]
 [0.91069704]
 [0.06526419]]
```



Autoencoder

- Both Models were trained under same conditions
- Both used MSE as Loss and SGD as optimizer
- TensorFlow had some difficulties with normal SGD so we add momentum term
- Both were trained for 150 epochs and with learning rate 0.01

P.O.C	Library	TensorFlow
Training Time (secs)	2400	1200
Loss	0.023402	0.0211

0.1

Challenges Faced

1- **Implementing Backpropagation:** Deriving and coding gradients for Dense layers and activation functions
2- **Training Stability:** Choosing an appropriate learning rate and number of epochs
3- **Autoencoder Reconstruction Quality:** Our First trials were giving blurry images because training with MSE is not ideal but after many iterations we succeeded in achieving desired output

Lessons Learned

1- **Understanding the Fundamentals Is Critical:** Implementing a neural network library from scratch highlighted the importance of understanding forward propagation, backpropagation, and gradient descent
2- **Choosing Appropriate Activation and loss Functions**: the choice of activation function greatly differs according to application and also loss functions are application dependent as we saw earlier it performed well for the XOR problem but not quite for the Autoencoder while Cross Entropy was better

Git-Hub Repo

https://github.com/Youssef-Ahmed73/Neural-Network-library-and-unsupervised-encoder-decoder-network-for-image-reconstruction.git