

Introduction to Neural Network & Deep Learning

Course Code: MEAD-654

Lab Practical File

Submitted by

Shantanu Shukla

01211805424

in partial fulfillment for the award of the degree

of

Master of Technology (AI & DS)

batch of 2024 - 26 (2nd Semester)



Center for Development of Advanced Computing, Noida

Affiliated to Guru Gobind Singh Indraprastha University

INDEX

S. No	Title	Page No.	Signature
1	Lab-1: Python and Deep Learning Libraries Setup	3-5	
2	Lab-2: Introduction to NumPy for Deep Learning	6-7	
3	Lab-3: Implementing a Perceptron	8-9	
4	Lab-4: Adaline and Gradient Descent	10-11	
5	Lab-5: Building a Feedforward Neural Network (from scratch)	12-13	
6	Lab-6: Building an MLP using PyTorch	14-15	
7	Lab-7: Visualizing Gradient Descent	16-17	
8	Lab-8: Exploring Optimizers	18-20	
9	Lab-9: Understanding Convolution Operations	21-22	
10	Lab-10: Building a CNN with PyTorch	23-25	

Lab-1

Objective: Set up Python, Jupyter Notebook, and install popular deep learning libraries (NumPy, TensorFlow, PyTorch).

Tasks:

- Install Anaconda, create a virtual environment.
- Install TensorFlow, PyTorch, and Keras.
- Write a simple program to confirm installations.

Results:

a.

Installing Anaconda.

```
--> docker
bash: /root/Anaconda3-2024.10-1-Linux-x86_64.sh: No such file or directory
root@e072939b1233:/# bash Anaconda3-2024.10-1-Linux-x86_64.sh

Welcome to Anaconda3 2024.10-1

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
ANACONDA TERMS OF SERVICE
Please read these Terms of Service carefully before purchasing, using, accessing, or downloading any Anaconda Offerings (the "Offerings"). These Anaconda Terms of Service ("TOS") are between Anaconda, Inc. ("Anaconda") and you ("You"), the individual or entity acquiring and/or providing access to the Offerings. These TOS govern Your access, download, installation, or use of the Anaconda Offerings, which are provided to You in combination with the terms set forth in the applicable Offering Description, and are hereby incorporated into these TOS. Except where indicated otherwise, references to "You" shall include Your Users. You hereby acknowledge that these TOS are binding, and You affirm and signify your consent to these TOS by registering to, using, installing, downloading, or accessing the Anaconda Offerings effective as of the date of first registration, use, install, download or access, as applicable (the "Effective Date"). Capitalized definitions not otherwise defined herein are set forth in Section 15 (Definitions). If You do not agree to these Terms of Service, You must not register, use, install, download, or access the Anaconda Offerings.

1. ACCESS & USE
1.1 General License Grant. Subject to compliance with these TOS and any applicable Offering Description, Anaconda grants You a personal, non-exclusive, non-transferable, non-sublicensable, revocable, limited right to use the applicable Anaconda Offering strictly as detailed herein and as set forth in a relevant Offering Description. If You purchase a subscription to an Offering as set forth in a relevant Order, then the license grant(s) applicable to your access, download, installation, or use of a specific Anaconda Offering will be set forth in the relevant Offering Description and any definitive agreement which may be executed by you in writing or electronic in connection with your Order ("Custom Agreement"). License grants for specific Anaconda Offerings are set forth in the relevant Offering Description, if applicable.
1.2 License Restrictions. Unless expressly agreed by Anaconda, You may not: (a) Make, sell, resell, license, sublicense, distribute, rent, or lease any Offerings available to anyone other than You or Your Users, unless expressly stated otherwise in an Order, Custom Agreement or the Documentation or as otherwise expressly permitted in writing by Anaconda; (b) Use the Offerings to store or transmit infringing, libelous, or otherwise unlawful or tortious material, or to store or transmit material in violation of third-party privacy rights; (c) Use the Offerings or Third Party Services to store or transmit Malicious Code, or attempt to gain unauthorized access to any Offerings or Third Party Services or their related systems or networks; (d) Interfere with or disrupt the integrity or performance of any Offerings or Third Party Services, or third-party data contained therein; (e) Permit direct or indirect access to or use of any Offerings or Third Party Services in a way that circumvents a contractual usage limit, or use any Offerings to access, copy or use any Anaconda intellectual property except as permitted under these TOS, a Custom Agreement, an Order or the Documentation; (f) Modify, copy or create derivative works of the Offerings or any part, feature, function or user interface thereof except, and then solely to the extent that, such activity is required to be permitted under applicable law; (g) Copy Content except as permitted herein or in an Order, a Custom Agreement or the Documentation or republish any material portion of any Offering in a manner competitive with the offering by Anaconda, including republication on another website or redistribute or embed any or all Offerings in a commercial product for redistribution or resale; (h) Frame or Mirror any part of any Content or Offerings, except if and to the extent permitted in an applicable Custom Agreement or Order for your own Internal Use and as permitted in a Custom Agreement or Documentation; (i) Except and then solely to the extent required to be permitted by applicable law, copy, disassemble, reverse engineer, or decompile an Offering, or access an Offering to build a competitive service by copying or using similar ideas, features, functions or graphics of the Offering. You may not use any "deep-link", "page-scrape", "robot", "spider" or other automatic device, program, algorithm or methodology, or any similar or equivalent manual process, to access, acquire, copy or monitor any portion of our Offerings or Content. Anaconda reserves the right to end any such activity. If You would like to redistribute or embed any Offering in any product You are developing, please contact the Anaconda team for a third party redistribution commercial license.

2. USERS & LICENSING
2.1 Organizational Use. Your registration, download, use, installation, access, or enjoyment of all Anaconda Offerings on behalf of an organization that has two hundred (200) or more employees or contractors ("Organizational Use") requires a paid license of Anaconda Business or Anaconda Enterprise. For sake of clarity, use by government entities and nonprofit entities with over 200 employees or contractors is considered Organizational Use. Purchasing Starter tier license(s) does not satisfy the Organizational Use paid license requirement set forth in this Section 2.1. Educational Entities will be exempt from the paid license requirement, provided that the use of the Anaconda Offering(s) is solely limited to being used for a curriculum-based course. Anaconda reserves the right to monitor the registration, download, use, installation, access, or enjoyment of the Anaconda Offerings to ensure it is part of a curriculum.
2.2 Use by Authorized Users. Your "Authorized Users" are your employees, agents, and independent contractors (including outsourcing service providers) who you authorize to use the Anaconda Offering(s) on Your behalf for Your Internal Use, provided that You are responsible for: (a) ensuring that such Authorized Users comply with these TOS or an applicable Custom Agreement; and (b) any breach of these TOS by such Authorized Users.
2.3 Use by Your Affiliates. Your Affiliates may use the Anaconda Offering(s) on Your behalf for Your Internal Use only with prior written approval from Anaconda. Such Affiliate usage is limited to those Affiliates who were defined as such upon the Effective Date of these TOS. Usage by organizations who become Your Affiliates after the Effective Date may require a separate license, at Anaconda's discretion.
2.4 Licenses for Systems. For each End User Computing Device ("EUCD") (i.e. Laptops, desktop devices) one license covers one installation and a reasonable number of virtual installations on the EUCD (e.g. Docker, VirtualBox, Parallels, etc.). Any other installations, usage, deployments, or access must have an individual license per each additional usage.
```

Creating a Virtual Environment.

```

--> docker
(base) root@a07293901233:/code#
(base) root@a07293901233:/code# conda create -n myenv python=3.12
Channels:
- defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /root/anaconda3/envs/myenv

added / updated specs:
- python=3.12

The following packages will be downloaded:

package | build | size
-----|-----|-----
ca-certificates-2025.2.25 | h6a4398_0 | 129 KB
expat-2.7.1 | h6e678d5_0 | 182 KB
openssl-3.0.16 | h5eee18b_0 | 5.2 MB
pip-25.1 | pyhc872135_2 | 1.3 MB
python-3.12.9 | h514d398_0 | 34.2 MB
setuptools-78.1.1 | py312h6a4398_0 | 2.2 MB
tzdata-2025b | h04d1e81_0 | 116 KB
wheel-0.45.1 | py312h6a4398_0 | 147 KB
xz-5.6.4 | h5eee18b_1 | 567 KB
-----|-----|-----
Total: 44.5 MB

The following NEW packages will be INSTALLED:

_libgcc_mutex | pkgs/main/linux-64::_libgcc_mutex-0.1-main
_openmp_mutex | pkgs/main/linux-64::_openmp_mutex-5.1-1_gnu
bzip2 | pkgs/main/linux-64::bzip2-1.0.8-h5eee18b_6
ca-certificates | pkgs/main/linux-64::ca-certificates-2025.2.25-h6a4398_0
expat | pkgs/main/linux-64::expat-2.7.1-h6e678d5_0
ld_impl_linux-64 | pkgs/main/linux-64::ld_impl_linux-64-2.40-h12ee557_0
libffi | pkgs/main/linux-64::libffi-3.4.4-h6e678d5_1
libgcc-ng | pkgs/main/linux-64::libgcc-ng-11.2.0-h1234567_1
libgomp | pkgs/main/linux-64::libgomp-11.2.0-h1234567_1
libstdcxx-ng | pkgs/main/linux-64::libstdcxx-ng-11.2.0-h1234567_1
libuuid | pkgs/main/linux-64::libuuid-1.41.5-h5eee18b_0
ncurses | pkgs/main/linux-64::ncurses-6.4-h6e678d5_0
openssl | pkgs/main/linux-64::openssl-3.0.16-h5eee18b_0
pip | pkgs/main/noarch::pip-25.1-pyhc872135_2

```

Activating the Virtual Environment.

```

#
# To activate this environment, use
#
#     $ conda activate myenv
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) root@a07293901233:/code#
(base) root@a07293901233:/code# conda activate myenv
(myenv) root@a07293901233:/code#
(myenv) root@a07293901233:/code# python -V
Python 3.12.9
(myenv) root@a07293901233:/code#
(myenv) root@a07293901233:/code#

```

b. Installing TensorFlow, PyTorch, and Keras libraries.

```

-> docker
(myenv) root@07293901233:/code# conda install tensorflow pytorch keras
Channels:
- defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /root/anaconda3/envs/myenv

added / updated specs:
- keras
- pytorch
- tensorflow

The following packages will be downloaded:

package | build | size
-----|-----|-----
absl-py-2.1.0 | py312h06a4398_0 | 239 KB
astunparse-1.6.3 | py_0 | 17 KB
brotli-python-1.0.9 | py312h06a78d5_9 | 356 KB
certifi-2025.4.26 | py312h06a4398_0 | 156 KB
cuda-cudart-12.4.127 | h99ab3db_0 | 21 KB
cuda-cudart-linux-64-12.4.127 | hda81fba_0 | 207 KB
cuda-nvrtc-12.4.127 | h99ab3db_1 | 19.8 MB
cuda-nvtx-12.4.127 | h0a678d5_1 | 30 KB
cuda-version-12.4 | hbd4654_3 | 19 KB
cudnn-9.1.1.17 | cuda12_1 | 477.5 MB
deprecated-1.2.13 | py312h06a4398_0 | 17 KB
filelock-3.17.0 | py312h06a4398_0 | 36 KB
flatbuffers-24.3.25 | h0a678d5_0 | 1.7 MB
fsspec-2025.3.2 | py312ha3bba80_0 | 620 KB
gast-0.5.3 | pyhd3eb1b0_0 | 21 KB
gitlib-5.2.2 | h5ee18b_0 | 80 KB
gmp-6.3.0 | h0a678d5_0 | 608 KB
gmpy2-2.2.1 | py312h5ee18b_0 | 259 KB
google-pasta-0.2.0 | pyhd3eb1b0_0 | 46 KB
grpcio-1.71.0 | py312h06a78d5_0 | 1.2 MB
h5py-3.12.1 | py312h5842655_1 | 1.4 MB
hdf5-1.14.5 | h2b7332_2 | 5.9 MB
importlib-metadata-8.5.0 | py312h06a4398_0 | 52 KB
jinja2-3.1.6 | py312h06a4398_0 | 352 KB
keras-3.6.0 | py312h06a4398_0 | 2.9 MB
libabseil-20250127.0 | cxx17_h0a678d5_1 | 1.3 MB
libcublas-12.4.5.8 | h99ab3db_1 | 247.6 MB

```

```

-> docker

Proceed ([y]/n)? y

Downloading and Extracting Packages:
libtorch-2.5.1 | 480.6 MB | #####5 | 27%
cudnn-9.1.1.17 | 477.5 MB | #####2 | 27%
libcublas-12.4.5.8 | 247.6 MB | #####6 | 62%
libcufft-11.2.1.3 | 174.5 MB | ##### | 100%
tensorflow-base-2.18 | 157.1 MB | #####5 | 82%
nccl-2.21.5.1 | 126.1 MB | #####5 | 37%
libcuparse-12.3.1.1 | 120.0 MB | #####5 | 0%
magma-2.7.1 | 117.1 MB | | 0%
libcusolver-11.6.1.9 | 83.5 MB | | 0%
libcureand-10.3.5.147 | 42.4 MB | | 0%
pytorch-2.5.1 | 35.5 MB | | 0%
cuda-nvrtc-12.4.127 | 19.8 MB | | 0%
libnvjitlink-12.4.12 | 17.4 MB | | 0%
sympy-1.13.3 | 15.0 MB | | 0%
numpy-base-2.0.1 | 8.5 MB | | 0%
libgpc-1.71.0 | 8.4 MB | | 0%
hdf5-1.14.5 | 5.9 MB | | 0%
tensorboard-2.18.0 | 5.5 MB | | 0%
tensorflow-data-ser | 4.2 MB | | 0%
libprotobuf-5.29.3 | 3.5 MB | | 0%
networkx-3.4.2 | 3.1 MB | | 0%
keras-3.6.0 | 2.9 MB | | 0%
pygments-2.19.1 | 2.2 MB | | 0%
flatbuffers-24.3.25 | 1.7 MB | | 0%
h5py-3.12.1 | 1.4 MB | | 0%
sleef-3.5.1 | 1.3 MB | | 0%
libabseil-20250127.0 | 1.3 MB | | 0%
grpcio-1.71.0 | 1.2 MB | | 0%
libuv-1.48.0 | 950 KB | | 0%
apfr-4.2.1 | 821 KB | | 0%
fsspec-2025.3.2 | 620 KB | | 0%
rich-13.9.4 | 615 KB | | 0%
gmp-6.3.0 | 608 KB | | 0%
protobuf-5.29.3 | 592 KB | | 0%
libcurl-8.12.1 | 469 KB | | 0%
optree-0.14.1 | 412 KB | | 0%
werkzeug-3.1.3 | 409 KB | | 0%
brotli-python-1.0.9 | 356 KB | | 0%
jinja2-3.1.6 | 352 KB | | 0%
mkl-random-1.2.8 | 324 KB | | 0%
libssh2-1.11.1 | 308 KB | | 0%
ml_dtypes-0.5.1 | 299 KB | | 0%

```

c. Confirming library installations.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(bash - code)

(myenv) (base) root@07293901233:/code# python lab-1/main.py
2025-05-16 08:20:53.265039: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
TensorFlow version: 2.18.1
Keras version: 3.6.0
PyTorch version: 2.5.1
WARNING:tensorflow:From /code/lab-1/main.py:9: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in a future version.
Instructions for updating:
Use 'tf.config.list_physical_devices('GPU')' instead.
TensorFlow is using GPU: False
PyTorch is using GPU: False
(myenv) (base) root@07293901233:/code#

```

Lab-2

Objective: Introduction to NumPy for Deep Learning.

Tasks:

- a. Create and manipulate arrays.
- b. Perform matrix multiplication, transpose, and reshaping.

Results:

a. *Create and manipulate arrays.*

```
1. import numpy as np
2.
3. # From a Python list
4. a = np.array([1, 2, 3])
5. print(a) # [1 2 3]
6.
7. # 2D array
8. b = np.array([[1, 2, 3], [4, 5, 6]])
9. print(b)
10.
11. # Zeros, Ones, and Identity
12. np.zeros((2, 3)) # 2x3 array of zeros
13. np.ones((3, 2)) # 3x2 array of ones
14. np.eye(3) # 3x3 identity matrix
15.
16. # Range and linspace
17. np.arange(0, 10, 2) # [0 2 4 6 8]
18. np.linspace(0, 1, 5) # [0.  0.25 0.5  0.75 1.  ]
19.
20. a = np.array([[10, 20, 30], [40, 50, 60]])
21.
22. print(a[0, 1]) # 20
23. print(a[:, 1]) # Column: [20 50]
24. print(a[1, :]) # Row: [40 50 60]
```

```
• (.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-2/a.py
[1 2 3]
[[1 2 3]
 [4 5 6]]
20
[20 50]
[40 50 60]
```

b. *Perform matrix multiplication, transpose, and reshaping.*

```
1. import numpy as np
2.
3. # Step 1: Create two matrices for multiplication
4. A = np.array([[1, 2, 3], [4, 5, 6]])
5. B = np.array([[7, 8], [9, 10], [11, 12]])
6.
7. # Step 2: Matrix multiplication (A is 2x3, B is 3x2 -> result is 2x2)
8. C = A @ B
9. print("Matrix Multiplication (A @ B):\n", C)
10.
11. # Step 3: Transpose the result
12. C_T = C.T
13. print("\nTranspose of the result:\n", C_T)
14.
15. # Step 4: Reshape the transposed matrix (from 2x2 to 1x4)
16. C_resaped = C_T.reshape((1, 4))
17. print("\nReshaped Transposed Matrix (1x4):\n", C_resaped)
```

```
• (.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-2/b.py
Matrix Multiplication (A @ B):
[[ 58  64]
 [139 154]]

Transpose of the result:
[[ 58 139]
 [ 64 154]]

Reshaped Transposed Matrix (1x4):
[[ 58 139 64 154]]
```

Lab-3

Objective: Understand and implement a simple Perceptron model from scratch.

Tasks:

- a. Initialize weights and bias.
- b. Implement the activation function (sign function).
- c. Train the Perceptron on a linearly separable dataset.

Results:

```
1. import numpy as np
2.
3.
4. class Perceptron:
5.     def __init__(self, input_size, learning_rate=0.1, epochs=10):
6.         # a. Initialize weights and bias
7.         self.weights = np.zeros(input_size)
8.         self.bias = 0.0
9.         self.lr = learning_rate
10.        self.epochs = epochs
11.
12.        # b. Activation function: Sign
13.        def activation(self, x):
14.            return 1 if x >= 0 else -1
15.
16.        def predict(self, x):
17.            linear_output = np.dot(self.weights, x) + self.bias
18.            return self.activation(linear_output)
19.
20.        # c. Training
21.        def train(self, X, y):
22.            for epoch in range(1, self.epochs + 1):
23.                print(f"\nEpoch {epoch}")
24.                for xi, target in zip(X, y):
25.                    pred = self.predict(xi)
26.                    error = target - pred
27.                    self.weights += self.lr * error * xi
28.                    self.bias += self.lr * error
29.                print(f" Weights: {self.weights}, Bias: {self.bias}")
30.
31.
32. # Linearly separable dataset
33. X = np.array([[2, 1], [1, -1], [-1, -2], [-2, -1]])
34. y = np.array([1, 1, -1, -1]) # Labels: +1 or -1
35.
36. # Train
```



```
37. p = Perceptron(input_size=2, learning_rate=0.1, epochs=10)
38. p.train(X, y)
39.
40. # Final predictions
41. print("\nFinal predictions:")
42. for xi in X:
43.     print(f"{xi} => {p.predict(xi)}")
```

```
• (.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-3/perceptron.py
```

```
Epoch 1
Weights: [0.2 0.4], Bias: -0.2
```

```
Epoch 2
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 3
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 4
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 5
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 6
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 7
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 8
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 9
Weights: [0.4 0.2], Bias: 0.0
```

```
Epoch 10
Weights: [0.4 0.2], Bias: 0.0
```

```
Final predictions:
```

```
[2 1] => 1
```

```
[ 1 -1] => 1
```

```
[-1 -2] => -1
```

```
[-2 -1] => -1
```

Lab-4

Objective: Implement Adaptive Linear Neuron (Adaline) using gradient descent.

Tasks:

- a. Initialize weights and bias.
- b. Use a continuous loss function (MSE).
- c. Implement gradient descent for weight updates.

Results:

```
1. import numpy as np
2.
3.
4. class Adaline:
5.     def __init__(self, input_size, learning_rate=0.01, epochs=10):
6.         # a. Initialize weights and bias
7.         self.weights = np.zeros(input_size)
8.         self.bias = 0.0
9.         self.lr = learning_rate
10.        self.epochs = epochs
11.
12.        def net_input(self, X):
13.            return np.dot(X, self.weights) + self.bias
14.
15.        def activation(self, X):
16.            # For Adaline, activation is linear (identity function)
17.            return self.net_input(X)
18.
19.        def train(self, X, y):
20.            for epoch in range(1, self.epochs + 1):
21.                # c. Compute predictions and errors
22.                output = self.activation(X)
23.                errors = y - output
24.
25.                # b. Compute Mean Squared Error (MSE)
26.                mse = np.mean(errors**2)
27.
28.                # c. Gradient descent: update weights and bias
29.                self.weights += self.lr * np.dot(X.T, errors)
30.                self.bias += self.lr * errors.sum()
31.
32.                print(
33.                    f"Epoch {epoch}: MSE = {mse:.4f}, Weights = {self.weights},
Bias = {self.bias}"
34.                )
35.
```

```

36.     def predict(self, X):
37.         return np.where(self.activation(X) >= 0.0, 1, -1)
38.
39.
40. if __name__ == "__main__":
41.     # Simple dataset: linearly separable
42.     X = np.array([[1, 1], [2, 1], [1, -1], [-1, -2], [-2, -1]])
43.     y = np.array([1, 1, 1, -1, -1]) # Targets in {-1, +1}
44.
45.     model = Adaline(input_size=2, learning_rate=0.01, epochs=10)
46.     model.train(X, y)
47.
48.     # Predictions
49.     print("\nFinal predictions:")
50.     for xi in X:
51.         print(f"{xi} => {model.predict(xi)}")

```

```

• (.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-4/main.py
Epoch 1: MSE = 1.0000, Weights = [0.07 0.04], Bias = 0.01
Epoch 2: MSE = 0.7561, Weights = [0.1298 0.0728], Bias = 0.0196
Epoch 3: MSE = 0.5807, Weights = [0.180958 0.09958 ], Bias = 0.028777999999999998
Epoch 4: MSE = 0.4543, Weights = [0.22479004 0.12133168], Bias = 0.03752112
Epoch 5: MSE = 0.3628, Weights = [0.26240802 0.13888817], Bias = 0.0458237972
Epoch 6: MSE = 0.2963, Weights = [0.29475161 0.15294911], Bias = 0.053686290416
Epoch 7: MSE = 0.2479, Weights = [0.32261513 0.16410181], Bias = 0.06111344190184
Epoch 8: MSE = 0.2124, Weights = [0.34667022 0.17283902], Bias = 0.06811365469040641
Epoch 9: MSE = 0.1862, Weights = [0.36748502 0.17957396], Bias = 0.0746980502405094
Epoch 10: MSE = 0.1667, Weights = [0.38554025 0.18465291], Bias = 0.08087977679447263

Final predictions:
[1 1] => 1
[2 1] => 1
[ 1 -1] => 1
[-1 -2] => -1
[-2 -1] => -1

```

Lab-5

Objective: Manually build an MLP with one hidden layer.

Tasks:

- a. Initialize weights and biases.
- b. Implement forward propagation.
- c. Apply activation functions (ReLU, Sigmoid).
- d. Implement backpropagation for weight updates.

Results:

```
1. import numpy as np
2.
3.
4. def sigmoid(x):
5.     return 1 / (1 + np.exp(-x))
6.
7.
8. def sigmoid_derivative(x):
9.     sx = sigmoid(x)
10.    return sx * (1 - sx)
11.
12.
13. def relu(x):
14.    return np.maximum(0, x)
15.
16.
17. def relu_derivative(x):
18.    return (x > 0).astype(float)
19.
20.
21. class MLP:
22.    def __init__(self, input_size, hidden_size, learning_rate=0.1):
23.        # a. Initialize weights and biases
24.        self.w1 = np.random.randn(input_size, hidden_size) # (2, 2)
25.        self.b1 = np.zeros((1, hidden_size)) # (1, 2)
26.        self.w2 = np.random.randn(hidden_size, 1) # (2, 1)
27.        self.b2 = np.zeros((1, 1)) # (1, 1)
28.        self.lr = learning_rate
29.
30.    def forward(self, X):
31.        # b. Forward propagation
32.        self.z1 = np.dot(X, self.w1) + self.b1
33.        self.a1 = relu(self.z1) # c. ReLU activation in hidden layer
34.
```

```

35.         self.z2 = np.dot(self.a1, self.w2) + self.b2
36.         self.a2 = sigmoid(self.z2) # c. Sigmoid activation in output
37.
38.         return self.a2
39.
40.     def backward(self, X, y):
41.         # d. Backpropagation
42.
43.         # Output layer error
44.         output_error = self.a2 - y # dL/da2
45.         output_delta = output_error * sigmoid_derivative(self.z2)
46.
47.         # Hidden layer error
48.         hidden_error = np.dot(output_delta, self.w2.T)
49.         hidden_delta = hidden_error * relu_derivative(self.z1)
50.
51.         # Gradient descent updates
52.         self.w2 -= self.lr * np.dot(self.a1.T, output_delta)
53.         self.b2 -= self.lr * np.sum(output_delta, axis=0, keepdims=True)
54.
55.         self.w1 -= self.lr * np.dot(X.T, hidden_delta)
56.         self.b1 -= self.lr * np.sum(hidden_delta, axis=0, keepdims=True)
57.
58.     def train(self, X, y, epochs=1000):
59.         for epoch in range(1, epochs + 1):
60.             output = self.forward(X)
61.             self.backward(X, y)
62.             if epoch % 100 == 0:
63.                 loss = np.mean((y - output) ** 2)
64.                 print(f"Epoch {epoch}: Loss = {loss:.4f}")
65.
66.     def predict(self, X):
67.         output = self.forward(X)
68.         return (output > 0.5).astype(int)
69.
70.
71. if __name__ == "__main__":
72.     # Input features
73.     X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
74.
75.     # XOR-like targets
76.     y = np.array([[0], [1], [1], [0]])
77.
78.     mlp = MLP(input_size=2, hidden_size=2, learning_rate=0.1)
79.     mlp.train(X, y, epochs=1000)
80.
81.     print("\nPredictions:")
82.     for xi in X:
83.         print(f"{xi} => {mlp.predict(np.array([xi]))[0][0]}")

```

```

Epoch 700: Loss = 0.1672
Epoch 800: Loss = 0.1671
Epoch 900: Loss = 0.1670
Epoch 1000: Loss = 0.1670

```

```

Predictions:
[0 0] => 0
[0 1] => 0
[1 0] => 1
[1 1] => 0

```

Lab-6

Objective: Use PyTorch to quickly build an MLP.

Tasks:

- a. Use nn.Module to create a model.
- b. Train the model using an optimizer (SGD).
- c. Evaluate model performance.

Results:

```
1. import torch
2. import torch.nn as nn
3. import torch.optim as optim
4.
5.
6. # a. Define MLP using nn.Module
7. class MLP(nn.Module):
8.     def __init__(self, input_size=2, hidden_size=4):
9.         super(MLP, self).__init__()
10.        self.model = nn.Sequential(
11.            nn.Linear(input_size, hidden_size),
12.            nn.ReLU(),
13.            nn.Linear(hidden_size, 1),
14.            nn.Sigmoid(), # for binary classification
15.        )
16.
17.        def forward(self, x):
18.            return self.model(x)
19.
20.
21. # Example XOR data
22. X = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32)
23. y = torch.tensor([[0], [1], [1], [0]], dtype=torch.float32)
24.
25. # Model, loss, optimizer
26. model = MLP()
27. criterion = nn.BCELoss() # Binary Cross Entropy Loss
28. optimizer = optim.SGD(model.parameters(), lr=0.1)
29.
30. # b. Training loop
31. for epoch in range(1000):
32.     # Forward pass
33.     output = model(X)
34.     loss = criterion(output, y)
35.
36.     # Backward pass
37.     optimizer.zero_grad()
38.     loss.backward()
39.     optimizer.step()
```

```
40.
41.     if (epoch + 1) % 100 == 0:
42.         print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
43.
44. # c. Evaluation
45. with torch.no_grad():
46.     predictions = model(X)
47.     predicted_classes = (predictions >= 0.5).float()
48.     accuracy = (predicted_classes == y).float().mean()
49.     print("\nPredictions:", predicted_classes.view(-1).tolist())
50.     print(f"Accuracy: {accuracy.item() * 100:.2f}%")
```

```
(.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-6/main.py
Epoch 100, Loss: 0.6936
Epoch 200, Loss: 0.6934
Epoch 300, Loss: 0.6932
Epoch 400, Loss: 0.6932
Epoch 500, Loss: 0.6932
Epoch 600, Loss: 0.6932
Epoch 700, Loss: 0.6932
Epoch 800, Loss: 0.6932
Epoch 900, Loss: 0.6931
Epoch 1000, Loss: 0.6931

Predictions: [1.0, 1.0, 0.0, 0.0]
Accuracy: 50.00%
```

Lab-7

Objective: Understand gradient descent with visualization.

Tasks:

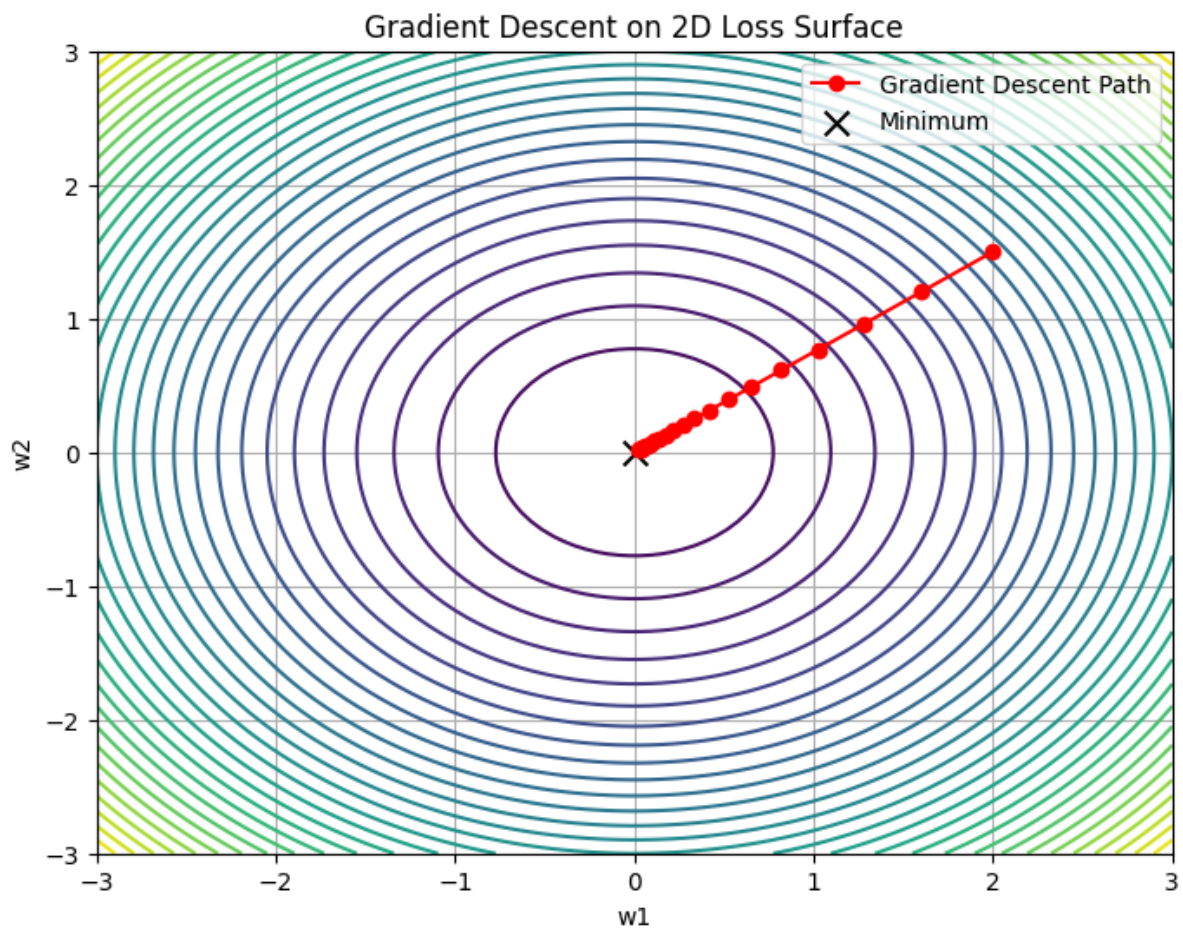
- a. Plot a 2D loss surface.
- b. Visualize gradient descent steps.

Results:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4.
5. # a. Define 2D quadratic loss surface:  $f(w) = w_1^2 + w_2^2$ 
6. def loss(w):
7.     return w[0] ** 2 + w[1] ** 2
8.
9.
10. def grad(w):
11.     return 2 * w # gradient of the loss
12.
13.
14. # Gradient Descent function
15. def gradient_descent(start, lr=0.1, steps=20):
16.     path = [start]
17.     w = start.copy()
18.
19.     for _ in range(steps):
20.         g = grad(w)
21.         w = w - lr * g
22.         path.append(w.copy())
23.
24.     return np.array(path)
25.
26.
27. # Generate path
28. start = np.array([2.0, 1.5]) # starting point
29. trajectory = gradient_descent(start, lr=0.1, steps=20)
30.
31. # b. Plot the loss surface and the path
32. w1 = np.linspace(-3, 3, 100)
33. w2 = np.linspace(-3, 3, 100)
34. W1, W2 = np.meshgrid(w1, w2)
35. Z = W1**2 + W2**2
36.
```



```
37. plt.figure(figsize=(8, 6))
38. plt.contour(W1, W2, Z, levels=30, cmap="viridis")
39. plt.plot(
40.     trajectory[:, 0], trajectory[:, 1], "o-", color="red", label="Gradient
Gradient Descent Path"
41. )
42. plt.scatter(0, 0, c="black", marker="x", s=100, label="Minimum")
43. plt.title("Gradient Descent on 2D Loss Surface")
44. plt.xlabel("w1")
45. plt.ylabel("w2")
46. plt.legend()
47. plt.grid(True)
48. plt.show()
```



Lab-8

Objective: Explore different optimization algorithms (SGD, Adam, RMSprop).

Tasks:

- a. Implement each optimizer manually.
- b. Compare their convergence on a sample problem.

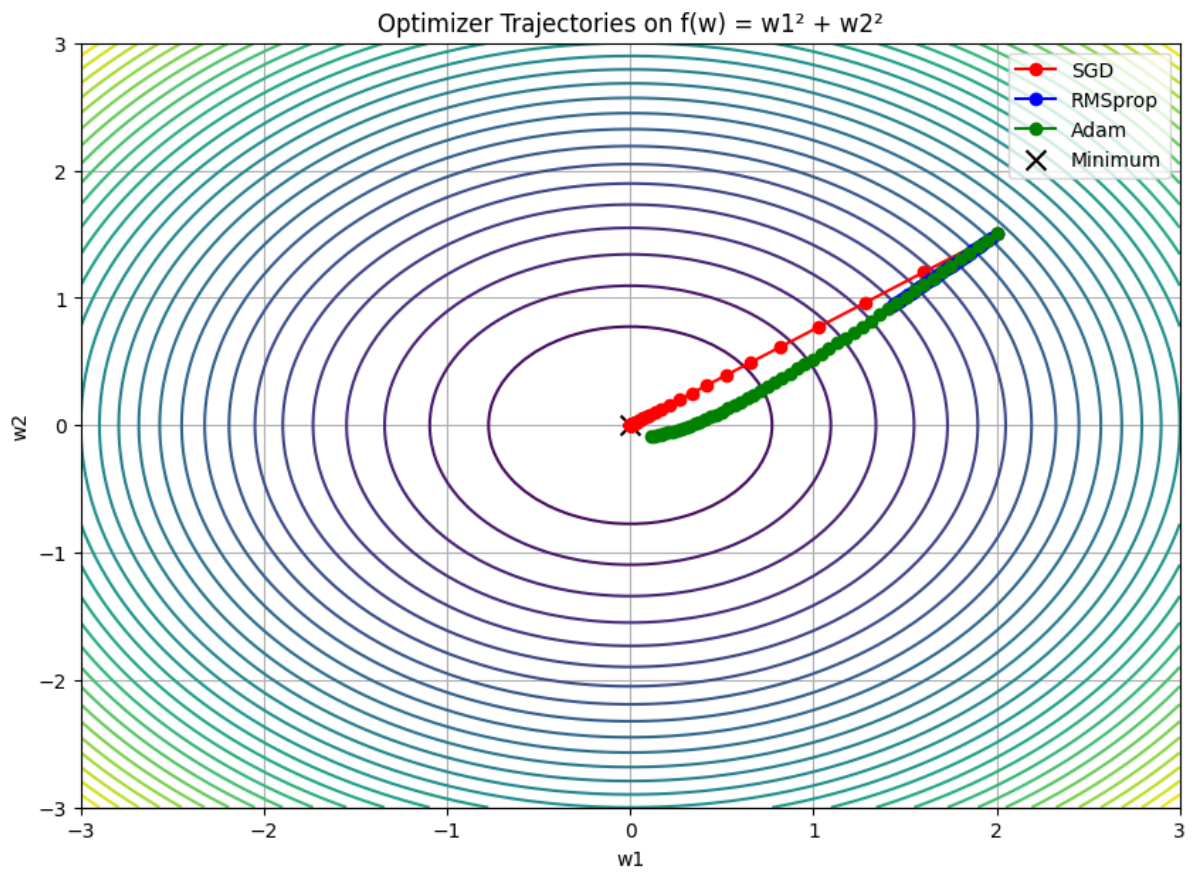
Results:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4.
5. # Loss and gradient
6. def loss(w):
7.     return w[0] ** 2 + w[1] ** 2
8.
9.
10. def grad(w):
11.     return 2 * w
12.
13.
14. # SGD optimizer
15. def sgd(w, lr):
16.     return w - lr * grad(w)
17.
18.
19. # RMSprop optimizer
20. def rmsprop(w, lr, grad_func, cache, beta=0.9, epsilon=1e-8):
21.     g = grad_func(w)
22.     cache = beta * cache + (1 - beta) * g**2
23.     w -= lr * g / (np.sqrt(cache) + epsilon)
24.     return w, cache
25.
26.
27. # Adam optimizer
28. def adam(w, lr, grad_func, m, v, t, beta1=0.9, beta2=0.999, epsilon=1e-8):
29.     g = grad_func(w)
30.     m = beta1 * m + (1 - beta1) * g
31.     v = beta2 * v + (1 - beta2) * g**2
32.
33.     m_hat = m / (1 - beta1**t)
34.     v_hat = v / (1 - beta2**t)
35.
36.     w -= lr * m_hat / (np.sqrt(v_hat) + epsilon)
37.     return w, m, v
38.
39.
```

```

40. def run_optimizer(name, steps=50, lr=0.1):
41.     w = np.array([2.0, 1.5])
42.     trajectory = [w.copy()]
43.
44.     # optimizer-specific variables
45.     cache = np.zeros_like(w)
46.     m, v = np.zeros_like(w), np.zeros_like(w)
47.
48.     for t in range(1, steps + 1):
49.         if name == "sgd":
50.             w = sgd(w, lr)
51.         elif name == "rmsprop":
52.             w, cache = rmsprop(w, lr, grad, cache)
53.         elif name == "adam":
54.             w, m, v = adam(w, lr, grad, m, v, t)
55.         trajectory.append(w.copy())
56.
57.     return np.array(trajectory)
58.
59.
60. # Generate trajectories
61. traj_sgd = run_optimizer("sgd", lr=0.1)
62. traj_rmsprop = run_optimizer("rmsprop", lr=0.01)
63. traj_adam = run_optimizer("adam", lr=0.05)
64.
65. # Plot loss surface
66. w1 = np.linspace(-3, 3, 100)
67. w2 = np.linspace(-3, 3, 100)
68. W1, W2 = np.meshgrid(w1, w2)
69. Z = W1**2 + W2**2
70.
71. plt.figure(figsize=(10, 7))
72. plt.contour(W1, W2, Z, levels=30, cmap="viridis")
73. plt.plot(traj_sgd[:, 0], traj_sgd[:, 1], "o-", label="SGD", color="red")
74. plt.plot(traj_rmsprop[:, 0], traj_rmsprop[:, 1], "o-", label="RMSprop",
75. color="blue")
76. plt.plot(traj_adam[:, 0], traj_adam[:, 1], "o-", label="Adam", color="green")
77. plt.scatter(0, 0, c="black", marker="x", s=100, label="Minimum")
78. plt.title("Optimizer Trajectories on  $f(w) = w_1^2 + w_2^2$ ")
79. plt.xlabel("w1")
80. plt.ylabel("w2")
81. plt.legend()
82. plt.grid(True)
83. plt.show()

```



Lab-9

Objective: Manually implement convolution and pooling.

Tasks:

- a. Create a 2D image matrix.
- b. Apply convolution with a kernel.

Results:

```
1. import numpy as np
2.
3. # a. Sample 5x5 grayscale image (values from 0 to 255)
4. image = np.array(
5.     [
6.         [10, 50, 80, 60, 20],
7.         [30, 100, 200, 150, 30],
8.         [50, 120, 255, 180, 40],
9.         [20, 90, 160, 140, 20],
10.        [10, 60, 70, 50, 10],
11.    ]
12. )
13.
14. # b. Example 3x3 kernel (edge detector / sharpening filter)
15. kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
16.
17.
18. def convolve2d(image, kernel):
19.     """Manual 2D convolution without padding"""
20.     image_h, image_w = image.shape
21.     kernel_h, kernel_w = kernel.shape
22.     output_h = image_h - kernel_h + 1
23.     output_w = image_w - kernel_w + 1
24.
25.     output = np.zeros((output_h, output_w))
26.
27.     for i in range(output_h):
28.         for j in range(output_w):
29.             region = image[i : i + kernel_h, j : j + kernel_w]
30.             output[i, j] = np.sum(region * kernel)
31.
32.     return output
33.
34.
35. # Perform convolution
36. convolved = convolve2d(image, kernel)
37.
38. # Display results
39. print("Original Image:\n", image)
```

```
40. print("\nKernel:\n", kernel)
41. print("\nConvolved Output:\n", convolved.astype(int))
```

```
(.venv) vscode → /workspaces/cdac-labwork/sem-2/NN&DL (main) $ uv run lab-9/main.py
Original Image:
[[ 10  50  80  60  20]
 [ 30 100 200 150  30]
 [ 50 120 255 180  40]
 [ 20  90 160 140  20]
 [ 10  60  70  50  10]]

Kernel:
[[-1 -1 -1]
 [-1  8 -1]
 [-1 -1 -1]]

Convolved Output:
[[  5 605 335]
 [ 55 900 445]
 [-25 315 335]]
```

Lab-10

Objective: Implement a simple CNN model.

Tasks:

- a. Use nn.Conv2d for convolution.
- b. Train on the MNIST dataset.

Results:

```

1. import matplotlib.pyplot as plt
2. import numpy as np
3. import torch
4. import torch.nn as nn
5. import torch.optim as optim
6. import torchvision
7. import torchvision.transforms as transforms
8. from torch.utils.data import DataLoader
9.
10. # Transform to Tensor and normalize to [0, 1]
11. transform = transforms.Compose(
12.     [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
13. )
14.
15. # Download and load training and test sets
16. train_set = torchvision.datasets.MNIST(
17.     root="./data", train=True, download=True, transform=transform
18. )
19. test_set = torchvision.datasets.MNIST(
20.     root="./data", train=False, download=True, transform=transform
21. )
22.
23. train_loader = DataLoader(train_set, batch_size=64, shuffle=True)
24. test_loader = DataLoader(test_set, batch_size=64, shuffle=False)
25.
26.
27. class SimpleCNN(nn.Module):
28.     def __init__(self):
29.         super(SimpleCNN, self).__init__()
30.         self.conv1 = nn.Conv2d(
31.             in_channels=1, out_channels=8, kernel_size=3
32.         ) # 1x28x28 -> 8x26x26
33.         self.pool = nn.MaxPool2d(kernel_size=2, stride=2) # 8x26x26 ->
8x13x13
34.         self.fc1 = nn.Linear(8 * 13 * 13, 10) # Fully connected layer for 10
classes
35.
36.     def forward(self, x):
37.         x = self.pool(torch.relu(self.conv1(x)))

```

```

38.         x = x.view(-1, 8 * 13 * 13) # Flatten
39.         x = self.fc1(x)
40.         return x
41.
42.
43. device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
44. model = SimpleCNN().to(device)
45. criterion = nn.CrossEntropyLoss()
46. optimizer = optim.Adam(model.parameters(), lr=0.001)
47.
48. # Training loop
49. for epoch in range(5):
50.     running_loss = 0.0
51.     for images, labels in train_loader:
52.         images, labels = images.to(device), labels.to(device)
53.
54.         optimizer.zero_grad()
55.         outputs = model(images)
56.         loss = criterion(outputs, labels)
57.         loss.backward()
58.         optimizer.step()
59.
60.         running_loss += loss.item()
61.
62.     print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")
63.
64. correct = 0
65. total = 0
66. model.eval()
67.
68. with torch.no_grad():
69.     for images, labels in test_loader:
70.         images, labels = images.to(device), labels.to(device)
71.         outputs = model(images)
72.         _, predicted = torch.max(outputs, 1)
73.         total += labels.size(0)
74.         correct += (predicted == labels).sum().item()
75.
76. accuracy = 100 * correct / total
77. print(f"Test Accuracy: {accuracy:.2f}%")
78.
79.
80. # Helper: Unnormalize for plotting
81. def imshow(img):
82.     img = img * 0.5 + 0.5 # unnormalize from [-1, 1] to [0, 1]
83.     npimg = img.numpy()
84.     plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap="gray")
85.     plt.axis("off")
86.
87.
88. # Get a batch of test images
89. dataiter = iter(test_loader)
90. images, labels = next(dataiter)
91. images, labels = images.to(device), labels.to(device)
92.
93. # Predict
94. model.eval()
95. with torch.no_grad():
96.     outputs = model(images)

```



```
97.     _, predicted = torch.max(outputs, 1)
98.
99. # Show images
100. plt.figure(figsize=(10, 5))
101. imshow(torchvision.utils.make_grid(images.cpu()[:8], nrow=8))
102. plt.title("Sample Predictions")
103. plt.show()
104.
105. # Print labels below the images
106. print("Ground Truth: ", " ".join(f"{labels[j].item()}" for j in range(8)))
107. print("Predicted:     ", " ".join(f"{predicted[j].item()}" for j in range(8)))
```

```
... Epoch 1, Loss: 0.3204
Epoch 2, Loss: 0.1057
Epoch 3, Loss: 0.0789
Epoch 4, Loss: 0.0655
Epoch 5, Loss: 0.0589
Test Accuracy: 97.92%
```

Sample Predictions



```
.. Ground Truth:  7 2 1 0 4 1 4 9
Predicted:       7 2 1 0 4 1 4 9
```