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

Lab-1

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

Tasks:

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

Results:

a.

Installing Anaconda.

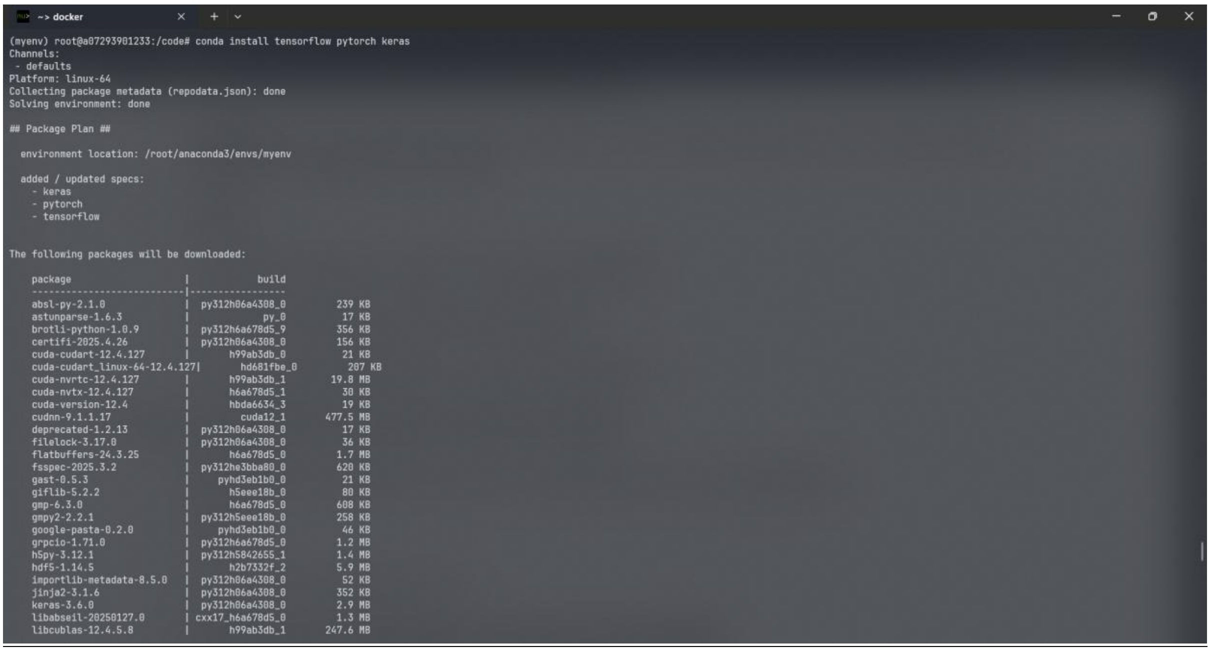
Creating a Virtual Environment.

```
~> 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
    ---------------------------|-----------------
    ca-certificates-2025.2.25  |       h06a4308_0         129 KB
    expat-2.7.1                |       h6a678d5_0         182 KB
    openssl-3.0.16             |       h5eee18b_0         5.2 MB
    pip-25.1                   |     pyhc872135_2         1.3 MB
    python-3.12.9              |       h5148396_0        34.7 MB
    setuptools-78.1.1          |  py312h06a4308_0         2.2 MB
    tzdata-2025b               |       h04d1e81_0         116 KB
    wheel-0.45.1               |  py312h06a4308_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-h06a4308_0
  expat              pkgs/main/linux-64::expat-2.7.1-h6a678d5_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-h6a678d5_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-h6a678d5_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@a07293901233:/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
    ---------------------------|-----------------
    absl-py-2.1.0              |  py312h06a4308_0         239 KB
    astunparse-1.6.3           |             py_0          17 KB
    brotli-python-1.0.9        |  py312h6a678d5_9         356 KB
    certifi-2025.4.26          |  py312h06a4308_0         156 KB
    cuda-cudart-12.4.127       |       h99ab3db_0          21 KB
    cuda-cudart_linux-64-12.4.127|       hd681fbe_0         207 KB
    cuda-nvrtc-12.4.127        |       h99ab3db_1        19.8 MB
    cuda-nvtx-12.4.127         |       h6a678d5_1          30 KB
    cuda-version-12.4          |       hbda6634_3          19 KB
    cudnn-9.1.1.17             |         cuda12_1       477.5 MB
    deprecated-1.2.13          |  py312h06a4308_0          17 KB
    filelock-3.17.0            |  py312h06a4308_0          36 KB
    flatbuffers-24.3.25        |       h6a678d5_0         1.7 MB
    fsspec-2025.3.2            |  py312he3bba80_0         620 KB
    gast-0.5.3                 |     pyhd3eb1b0_0          21 KB
    giflib-5.2.2               |       h5eee18b_0          80 KB
    gmp-6.3.0                  |       h6a678d5_0         608 KB
    gmpy2-2.2.1                |  py312h5eee18b_0         258 KB
    google-pasta-0.2.0         |     pyhd3eb1b0_0          46 KB
    grpcio-1.71.0              |  py312h6a678d5_0         1.2 MB
    h5py-3.12.1                |  py312h5842655_1         1.4 MB
    hdf5-1.14.5                |       h2b7332f_2         5.9 MB
    importlib-metadata-8.5.0   |  py312h06a4308_0          52 KB
    jinja2-3.1.6               |  py312h06a4308_0         352 KB
    keras-3.6.0                |  py312h06a4308_0         2.9 MB
    libabseil-20250127.0       | cxx17_h6a678d5_0         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  |   |  29%
cudnn-9.1.1.17       | 477.5 MB  |   |  27%
libcublas-12.4.5.8   | 247.6 MB  |   |  62%
libcufft-11.2.1.3    | 174.5 MB  |   | 100%
tensorflow-base-2.18 | 157.1 MB  |   |  82%
nccl-2.21.5.1        | 126.1 MB  |   |  37%
libcusparse-12.3.1.1 | 120.0 MB  |   |   0%
magma-2.7.1          | 117.1 MB  |   |   0%
libcusolver-11.6.1.9 | 83.5 MB   |   |   0%
libcurand-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%
libgrpc-1.71.0       | 8.4 MB    |   |   0%
hdf5-1.14.5          | 5.9 MB    |   |   0%
tensorboard-2.18.0   | 5.5 MB    |   |   0%
tensorboard-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%
mpfr-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@a07293901233:/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@a07293901233:/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.*

```python
import numpy as np

# From a Python list
a = np.array([1, 2, 3])
print(a)  # [1 2 3]

# 2D array
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)

# Zeros, Ones, and Identity
np.zeros((2, 3))  # 2x3 array of zeros
np.ones((3, 2))  # 3x2 array of ones
np.eye(3)  # 3x3 identity matrix

# Range and linspace
np.arange(0, 10, 2)  # [0 2 4 6 8]
np.linspace(0, 1, 5)  # [0.   0.25 0.5  0.75 1.  ]

a = np.array([[10, 20, 30], [40, 50, 60]])

print(a[0, 1])  # 20
print(a[:, 1])  # Column: [20 50]
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.*

```python
import numpy as np

# Step 1: Create two matrices for multiplication
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[7, 8], [9, 10], [11, 12]])

# Step 2: Matrix multiplication (A is 2x3, B is 3x2 -> result is 2x2)
C = A @ B
print("Matrix Multiplication (A @ B):\n", C)

# Step 3: Transpose the result
C_T = C.T
print("\nTranspose of the result:\n", C_T)

# Step 4: Reshape the transposed matrix (from 2x2 to 1x4)
C_reshaped = C_T.reshape((1, 4))
print("\nReshaped Transposed Matrix (1x4):\n", C_reshaped)
```

```
(.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:

```python
import numpy as np


class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=10):
        # a. Initialize weights and bias
        self.weights = np.zeros(input_size)
        self.bias = 0.0
        self.lr = learning_rate
        self.epochs = epochs

    # b. Activation function: Sign
    def activation(self, x):
        return 1 if x >= 0 else -1

    def predict(self, x):
        linear_output = np.dot(self.weights, x) + self.bias
        return self.activation(linear_output)

    # c. Training
    def train(self, X, y):
        for epoch in range(1, self.epochs + 1):
            print(f"\nEpoch {epoch}")
            for xi, target in zip(X, y):
                pred = self.predict(xi)
                error = target - pred
                self.weights += self.lr * error * xi
                self.bias += self.lr * error
            print(f" Weights: {self.weights}, Bias: {self.bias}")


# Linearly separable dataset
X = np.array([[2, 1], [1, -1], [-1, -2], [-2, -1]])
y = np.array([1, 1, -1, -1])  # Labels: +1 or -1

# Train
```

```python
p = Perceptron(input_size=2, learning_rate=0.1, epochs=10)
p.train(X, y)

# Final predictions
print("\nFinal predictions:")
for xi in X:
    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:

```
import numpy as np


class Adaline:
    def __init__(self, input_size, learning_rate=0.01, epochs=10):
        # a. Initialize weights and bias
        self.weights = np.zeros(input_size)
        self.bias = 0.0
        self.lr = learning_rate
        self.epochs = epochs

    def net_input(self, X):
        return np.dot(X, self.weights) + self.bias

    def activation(self, X):
        # For Adaline, activation is linear (identity function)
        return self.net_input(X)

    def train(self, X, y):
        for epoch in range(1, self.epochs + 1):
            # c. Compute predictions and errors
            output = self.activation(X)
            errors = y - output

            # b. Compute Mean Squared Error (MSE)
            mse = np.mean(errors**2)

            # c. Gradient descent: update weights and bias
            self.weights += self.lr * np.dot(X.T, errors)
            self.bias += self.lr * errors.sum()

            print(
                f"Epoch {epoch}: MSE = {mse:.4f}, Weights = {self.weights}, Bias = {self.bias}"
            )

```

```python
    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, -1)


if __name__ == "__main__":
    # Simple dataset: linearly separable
    X = np.array([[1, 1], [2, 1], [1, -1], [-1, -2], [-2, -1]])
    y = np.array([1, 1, 1, -1, -1])  # Targets in {-1, +1}

    model = Adaline(input_size=2, learning_rate=0.01, epochs=10)
    model.train(X, y)

    # Predictions
    print("\nFinal predictions:")
    for xi in X:
        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:

```
import numpy as np


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def sigmoid_derivative(x):
    sx = sigmoid(x)
    return sx * (1 - sx)


def relu(x):
    return np.maximum(0, x)


def relu_derivative(x):
    return (x > 0).astype(float)


class MLP:
    def __init__(self, input_size, hidden_size, learning_rate=0.1):
        # a. Initialize weights and biases
        self.w1 = np.random.randn(input_size, hidden_size)  # (2, 2)
        self.b1 = np.zeros((1, hidden_size))  # (1, 2)
        self.w2 = np.random.randn(hidden_size, 1)  # (2, 1)
        self.b2 = np.zeros((1, 1))  # (1, 1)
        self.lr = learning_rate

    def forward(self, X):
        # b. Forward propagation
        self.z1 = np.dot(X, self.w1) + self.b1
        self.a1 = relu(self.z1)  # c. ReLU activation in hidden layer

```

```python
        self.z2 = np.dot(self.a1, self.w2) + self.b2
        self.a2 = sigmoid(self.z2)  # c. Sigmoid activation in output

        return self.a2

    def backward(self, X, y):
        # d. Backpropagation

        # Output layer error
        output_error = self.a2 - y  # dL/da2
        output_delta = output_error * sigmoid_derivative(self.z2)

        # Hidden layer error
        hidden_error = np.dot(output_delta, self.w2.T)
        hidden_delta = hidden_error * relu_derivative(self.z1)

        # Gradient descent updates
        self.w2 -= self.lr * np.dot(self.a1.T, output_delta)
        self.b2 -= self.lr * np.sum(output_delta, axis=0, keepdims=True)

        self.w1 -= self.lr * np.dot(X.T, hidden_delta)
        self.b1 -= self.lr * np.sum(hidden_delta, axis=0, keepdims=True)

    def train(self, X, y, epochs=1000):
        for epoch in range(1, epochs + 1):
            output = self.forward(X)
            self.backward(X, y)
            if epoch % 100 == 0:
                loss = np.mean((y - output) ** 2)
                print(f"Epoch {epoch}: Loss = {loss:.4f}")

    def predict(self, X):
        output = self.forward(X)
        return (output > 0.5).astype(int)


if __name__ == "__main__":
    # Input features
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

    # XOR-like targets
    y = np.array([[0], [1], [1], [0]])

    mlp = MLP(input_size=2, hidden_size=2, learning_rate=0.1)
    mlp.train(X, y, epochs=1000)

    print("\nPredictions:")
    for xi in X:
        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:

```
import torch
import torch.nn as nn
import torch.optim as optim


# a. Define MLP using nn.Module
class MLP(nn.Module):
    def __init__(self, input_size=2, hidden_size=4):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 1),
            nn.Sigmoid(),  # for binary classification
        )

    def forward(self, x):
        return self.model(x)


# Example XOR data
X = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32)
y = torch.tensor([[0], [1], [1], [0]], dtype=torch.float32)

# Model, loss, optimizer
model = MLP()
criterion = nn.BCELoss()  # Binary Cross Entropy Loss
optimizer = optim.SGD(model.parameters(), lr=0.1)

# b. Training loop
for epoch in range(1000):
    # Forward pass
    output = model(X)
    loss = criterion(output, y)

    # Backward pass
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```python

    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

# c. Evaluation
with torch.no_grad():
    predictions = model(X)
    predicted_classes = (predictions >= 0.5).float()
    accuracy = (predicted_classes == y).float().mean()
    print("\nPredictions:", predicted_classes.view(-1).tolist())
    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:

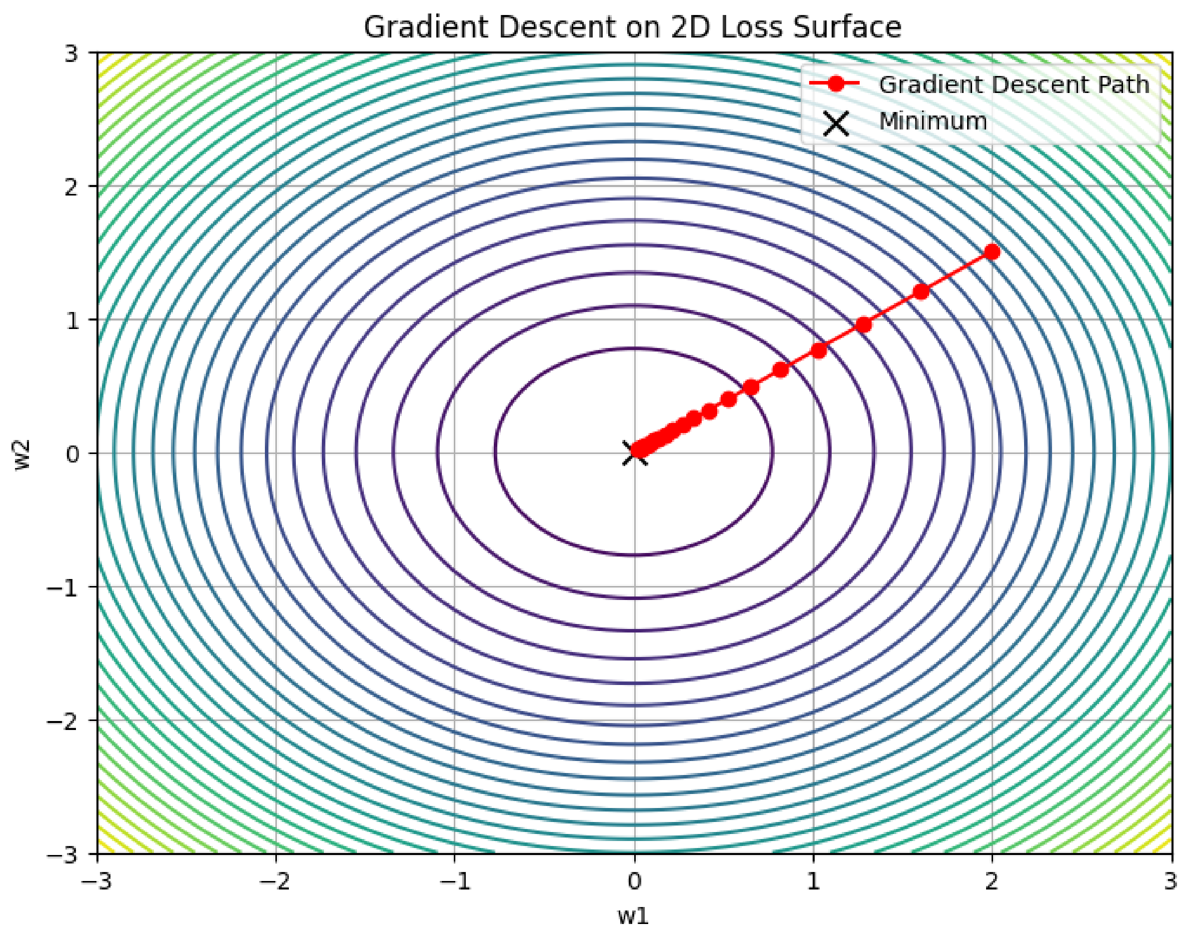
a. Plot a 2D loss surface.

b. Visualize gradient descent steps.

Results:

```
import numpy as np
import matplotlib.pyplot as plt


# a. Define 2D quadratic loss surface: f(w) = w1² + w2²
def loss(w):
    return w[0] ** 2 + w[1] ** 2


def grad(w):
    return 2 * w  # gradient of the loss


# Gradient Descent function
def gradient_descent(start, lr=0.1, steps=20):
    path = [start]
    w = start.copy()

    for _ in range(steps):
        g = grad(w)
        w = w - lr * g
        path.append(w.copy())

    return np.array(path)


# Generate path
start = np.array([2.0, 1.5])  # starting point
trajectory = gradient_descent(start, lr=0.1, steps=20)

# b. Plot the loss surface and the path
w1 = np.linspace(-3, 3, 100)
w2 = np.linspace(-3, 3, 100)
W1, W2 = np.meshgrid(w1, w2)
Z = W1**2 + W2**2

```

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



Lab-8

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

Tasks:

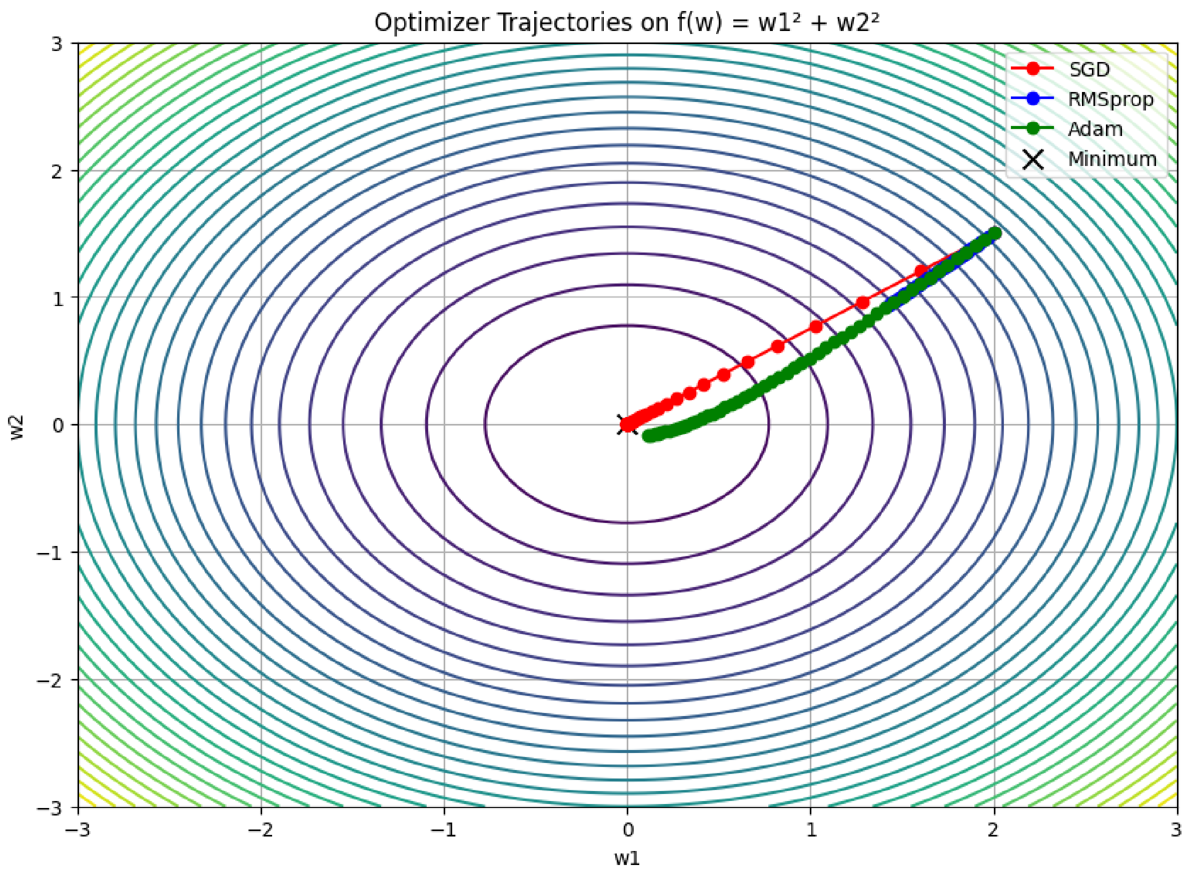
a. Implement each optimizer manually.

b. Compare their convergence on a sample problem.

Results:

```
import numpy as np
import matplotlib.pyplot as plt


# Loss and gradient
def loss(w):
    return w[0] ** 2 + w[1] ** 2


def grad(w):
    return 2 * w


# SGD optimizer
def sgd(w, lr):
    return w - lr * grad(w)


# RMSprop optimizer
def rmsprop(w, lr, grad_func, cache, beta=0.9, epsilon=1e-8):
    g = grad_func(w)
    cache = beta * cache + (1 - beta) * g**2
    w -= lr * g / (np.sqrt(cache) + epsilon)
    return w, cache


# Adam optimizer
def adam(w, lr, grad_func, m, v, t, beta1=0.9, beta2=0.999, epsilon=1e-8):
    g = grad_func(w)
    m = beta1 * m + (1 - beta1) * g
    v = beta2 * v + (1 - beta2) * g**2

    m_hat = m / (1 - beta1**t)
    v_hat = v / (1 - beta2**t)

    w -= lr * m_hat / (np.sqrt(v_hat) + epsilon)
    return w, m, v


```

```python
def run_optimizer(name, steps=50, lr=0.1):
    w = np.array([2.0, 1.5])
    trajectory = [w.copy()]

    # optimizer-specific variables
    cache = np.zeros_like(w)
    m, v = np.zeros_like(w), np.zeros_like(w)

    for t in range(1, steps + 1):
        if name == "sgd":
            w = sgd(w, lr)
        elif name == "rmsprop":
            w, cache = rmsprop(w, lr, grad, cache)
        elif name == "adam":
            w, m, v = adam(w, lr, grad, m, v, t)
        trajectory.append(w.copy())

    return np.array(trajectory)


# Generate trajectories
traj_sgd = run_optimizer("sgd", lr=0.1)
traj_rmsprop = run_optimizer("rmsprop", lr=0.01)
traj_adam = run_optimizer("adam", lr=0.05)

# Plot loss surface
w1 = np.linspace(-3, 3, 100)
w2 = np.linspace(-3, 3, 100)
W1, W2 = np.meshgrid(w1, w2)
Z = W1**2 + W2**2

plt.figure(figsize=(10, 7))
plt.contour(W1, W2, Z, levels=30, cmap="viridis")
plt.plot(traj_sgd[:, 0], traj_sgd[:, 1], "o-", label="SGD", color="red")
plt.plot(traj_rmsprop[:, 0], traj_rmsprop[:, 1], "o-", label="RMSprop", color="blue")
plt.plot(traj_adam[:, 0], traj_adam[:, 1], "o-", label="Adam", color="green")
plt.scatter(0, 0, c="black", marker="x", s=100, label="Minimum")
plt.title("Optimizer Trajectories on f(w) = w1² + w2²")
plt.xlabel("w1")
plt.ylabel("w2")
plt.legend()
plt.grid(True)
plt.show()
```



Optimizer Trajectories on f(w) = w1² + w2²
SGD
RMSprop
Adam
Minimum
w1
w2
−3
−2
−1
0
1
2
3

Lab-9

Objective: Manually implement convolution and pooling.

Tasks:

a. Create a 2D image matrix.

b. Apply convolution with a kernel.

Results:

```
import numpy as np

# a. Sample 5x5 grayscale image (values from 0 to 255)
image = np.array(
    [
        [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],
    ]
)

# b. Example 3x3 kernel (edge detector / sharpening filter)
kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])


def convolve2d(image, kernel):
    """Manual 2D convolution without padding"""
    image_h, image_w = image.shape
    kernel_h, kernel_w = kernel.shape
    output_h = image_h - kernel_h + 1
    output_w = image_w - kernel_w + 1

    output = np.zeros((output_h, output_w))

    for i in range(output_h):
        for j in range(output_w):
            region = image[i : i + kernel_h, j : j + kernel_w]
            output[i, j] = np.sum(region * kernel)

    return output


# Perform convolution
convolved = convolve2d(image, kernel)

# Display results
print("Original Image:\n", image)
```

```
print("\nKernel:\n", kernel)
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.

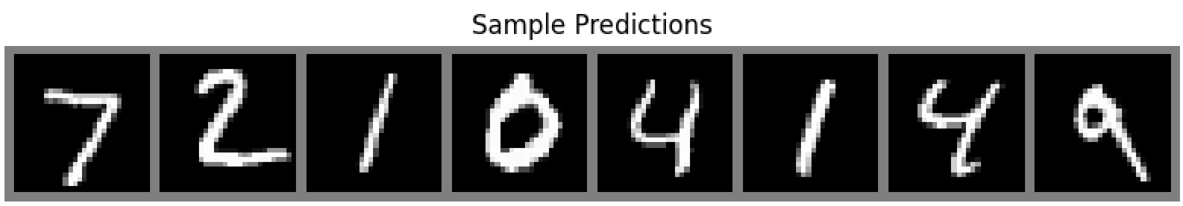
Results:

```
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# Transform to Tensor and normalize to [0, 1]
transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
)

# Download and load training and test sets
train_set = torchvision.datasets.MNIST(
    root="./data", train=True, download=True, transform=transform
)
test_set = torchvision.datasets.MNIST(
    root="./data", train=False, download=True, transform=transform
)

train_loader = DataLoader(train_set, batch_size=64, shuffle=True)
test_loader = DataLoader(test_set, batch_size=64, shuffle=False)


class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels=1, out_channels=8, kernel_size=3
        )  # 1x28x28 -> 8x26x26
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  # 8x26x26 -> 8x13x13
        self.fc1 = nn.Linear(8 * 13 * 13, 10)  # Fully connected layer for 10 classes

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
```

```
        x = x.view(-1, 8 * 13 * 13)  # Flatten
        x = self.fc1(x)
        return x


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(5):
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

correct = 0
total = 0
model.eval()

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy: {accuracy:.2f}%")


# Helper: Unnormalize for plotting
def imshow(img):
    img = img * 0.5 + 0.5  # unnormalize from [-1, 1] to [0, 1]
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap="gray")
    plt.axis("off")


# Get a batch of test images
dataiter = iter(test_loader)
images, labels = next(dataiter)
images, labels = images.to(device), labels.to(device)

# Predict
model.eval()
with torch.no_grad():
    outputs = model(images)
```

```
        _, predicted = torch.max(outputs, 1)

# Show images
plt.figure(figsize=(10, 5))
imshow(torchvision.utils.make_grid(images.cpu()[:8], nrow=8))
plt.title("Sample Predictions")
plt.show()

# Print labels below the images
print("Ground Truth: ", " ".join(f"{labels[j].item()}" for j in range(8)))
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%
```




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