Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Deep Learning and Reinforcement Learning

[BAI701]

Prepared by
Mr. Dharaneshkumar M L
Asst. Prof. Dept. of AI & DS,
CIT, Gubbi

Department of Artificial Intelligence and Data Science

VII Semester

Academic Year: 2025-26

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(**NAAC Accredited & ISO 9001:2015 Certified Institution**)

NH 206 (B.H. Road), Gubbi, Tumkur – 572216. Karnataka.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AN DATA SCIENCE

Syllabus

Sl.NO	**Experiments**
1	Design and implement a neural based network for generating word embedding for words in a document corpus
2	Write a program to demonstrate the working of a deep neural network for classification task.
3	Design and implement a Convolutional Neural Network(CNN) for classification of image dataset
4	Build and demonstrate an auto encoder network using neural layers for data compression on image dataset.
5	Design and implement a deep learning network for classification of textual documents.
6	Design and implement a deep learning network for forecasting time series data.
7	Write a program to enable pre-train models to classify a given image dataset
8	Simple Grid World Problem: Design a custom 2D grid world where the agent navigates from a start position to a goal, avoiding obstacles. Environment: Custom grid (easily implemented in Python)

Course outcomes (Course Skill Set):

At the end of the course, the student will be able to:

CO1: Demonstrate the implementation of deep learning techniques

CO2: Examine various deep learning techniques for solving the real world problems

CO3: Design and implement research-oriented scenario using deep learning techniques in a team

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%.The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE
(Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the practical component of the IPCC

15 marks for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.

On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.

- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks.**

- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks.**
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

How to Install Anaconda on windows 10/11

Using Anaconda (Recommended)

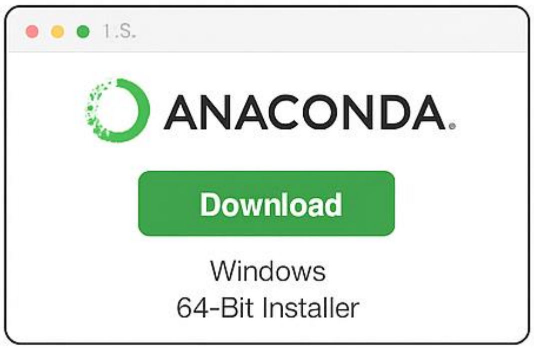
1. **Download Anaconda**
   - Go to https://www.anaconda.com/download.
   - Select the **Windows** version (64-bit installer).
2. **Run the Installer**
   - Double-click the downloaded `.exe` file.
   - Choose "Just Me" (recommended) or "All Users."
   - Keep the default installation path unless you prefer otherwise.
   - Check **"Add Anaconda to my PATH environment variable"** (optional but convenient).
3. **Launch Jupyter Notebook**
   - Open **Anaconda Navigator** from the Start Menu.
   - Click **"Launch"** under Jupyter Notebook.
   - It will open in your default browser at `http://localhost:8888`.

Jupyter Notebook in Windows

1 **Download Anaconda**

Visit www.anaconda.com /download.

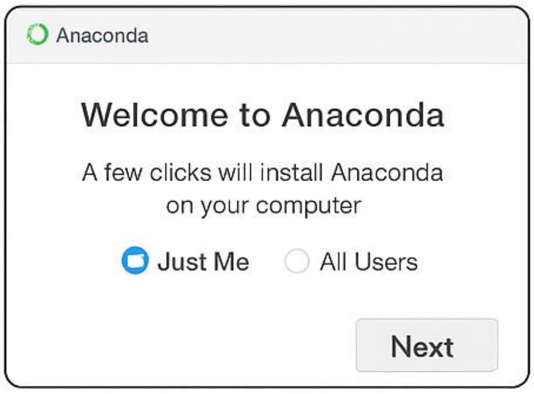
Select the Windows (64-bit) installer.



2 **Run the Installer**

Double-click the downloaded.,exe file.

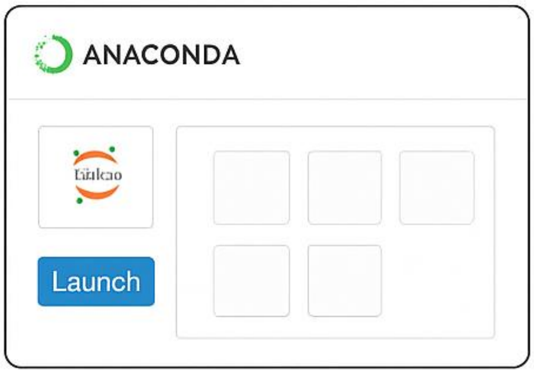
Click "Just Me" (recommended) or "All Users".



3 **Launch Jupyter Notebook**

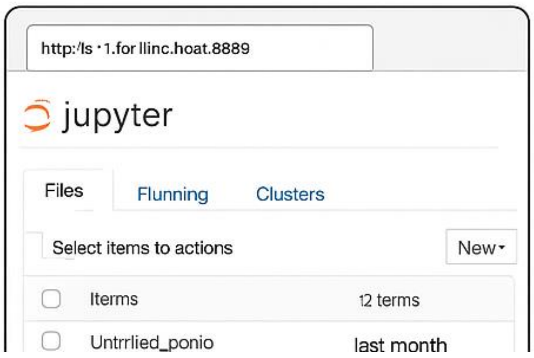
Open Anaconda Navigator from the Start Menu.

Click "Launch" under Jupyter Notebook.



4 **Open Jupyter Notebook**

Jupyter Notebook opens in your default browser at http://localhost:8888



Program 1

Design and implement a neural based network for generating word embedding for words in a document corpus.

Install Required Packages

```
pip install torch
```

(If pip is not recognized, use `python -m pip install torch`.)

Program:

```
import torch

import torch.nn as nn

import torch.optim as optim

from collections import Counter

import re

import random


# ------------------------------

# 1. Preprocess Corpus

# ------------------------------

def tokenize(text):

    # very simple tokenizer: lowercase + split on non-letters

    return re.findall(r"\b\w+\b", text.lower())


corpus = """Natural language processing enables computers to understand human language.

Word embeddings map words into continuous vector space.

Neural networks can be used to learn these embeddings from context words in a document corpus."""

tokens = tokenize(corpus)


# ------------------------------

# 2. Build Vocabulary

# ------------------------------

word_counts = Counter(tokens)

vocab = {w: i for i, (w, _) in enumerate(word_counts.items())}

id2word = {i: w for w, i in vocab.items()}

vocab_size = len(vocab)
```

```
# ------------------------------
# 3. Generate Training Data (Skip-gram)
# ------------------------------
def generate_pairs(tokens, window_size=2):
    pairs = []
    for i, center in enumerate(tokens):
        for j in range(max(0, i - window_size), min(len(tokens), i + window_size + 1)):
            if i != j:
                pairs.append((center, tokens[j]))
    return pairs

pairs = generate_pairs(tokens, window_size=2)
print("Sample training pairs:", pairs[:10])

# Convert to indices
training_data = [(vocab[c], vocab[o]) for c, o in pairs]

# ------------------------------
# 4. Define Neural Network
# ------------------------------
class Word2Vec(nn.Module):
    def __init__(self, vocab_size, embed_dim):
        super(Word2Vec, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_dim)  # input layer
        self.output = nn.Linear(embed_dim, vocab_size)         # output layer

    def forward(self, center_word):
        embed = self.embeddings(center_word)
        out = self.output(embed)
        return out

# ------------------------------
# 5. Training
# ------------------------------
embed_dim = 50
model = Word2Vec(vocab_size, embed_dim)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

for epoch in range(50):
    total_loss = 0
    for center, context in training_data:
        center_tensor = torch.tensor([center], dtype=torch.long)
        context_tensor = torch.tensor([context], dtype=torch.long)

        optimizer.zero_grad()
        output = model(center_tensor)
        loss = criterion(output, context_tensor)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if (epoch+1) % 10 == 0:
        print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}")

# ------------------------------
# 6. Extract Word Embeddings
# ------------------------------
embeddings = model.embeddings.weight.data
print("\nWord Embedding for 'language':\n", embeddings[vocab["language"]])
print("\nWord Embedding for 'neural':\n", embeddings[vocab["neural"]])
```

Output:

```
Sample training pairs: [('natural', 'language'), ('natural', 'processing'), ('lang
uage', 'natural'), ('language', 'processing'), ('language', 'enables'), ('processi
ng', 'natural'), ('processing', 'language'), ('processing', 'enables'), ('processi
ng', 'computers'), ('enables', 'language')]
Epoch 10, Loss: 266.1342
Epoch 20, Loss: 250.3976
Epoch 30, Loss: 239.7421
Epoch 40, Loss: 234.2845
Epoch 50, Loss: 230.9848

Word Embedding for 'language':
 tensor([ 0.0204, -0.1836,  0.8128, -0.0148, -0.0079,  0.3005, -0.0173,  0.1209,
         0.9695,  0.0854,  0.1376, -0.0599,  0.3838,  0.1456,  0.3343,  0.1069,
        -0.0225, -0.0558, -0.0046,  0.4430,  0.0055,  0.2052,  0.1331, -0.3018,
        -0.0151,  0.0913,  0.8319,  0.0721,  0.0844, -0.0092, -0.3637,  0.1117,
        -0.2003,  0.2453,  0.6428, -0.0277,  0.1391, -0.0239,  0.1692, -0.6957,
         1.1484, -0.0126, -0.1407,  0.7661,  0.2213, -0.3476,  0.0141, -0.3157,
         0.6018,  0.7495])

Word Embedding for 'neural':
 tensor([-1.5870,  1.3447, -0.8288,  0.2156, -0.9003, -0.8805,  0.2801, -0.3090,
         0.1337, -0.7787,  0.1426,  0.3447,  0.2340, -0.0515, -0.5118, -1.1024,
         0.6115, -0.3054, -0.5795, -1.1323, -0.6081,  0.8951, -0.1091,  0.3673,
         0.9300, -0.1212, -1.1510, -0.1241,  0.8780, -0.0572, -0.1193, -0.0332,
        -0.2150, -0.0204, -0.8725,  0.9793, -0.4665,  0.2905, -0.2161, -0.0386,
        -0.1379,  1.8547, -1.1795, -0.6961,  0.7195, -0.4361,  0.2295, -0.8763,
         0.0351,  0.5738])
```

Program 2

Write a program to demonstrate the working of a deep neural network for classification task.

Install Required Packages

```
pip install torch

pip install scikit-learn
```

Program:

```
import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


# ------------------------------

# 1. Load Dataset (Iris)

# ------------------------------

iris = load_iris()

X = iris.data   # features

y = iris.target # labels (0, 1, 2)


# Normalize features

scaler = StandardScaler()

X = scaler.fit_transform(X)


# Convert to torch tensors

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

# ------------------------------
# 2. Define Deep Neural Network
# ------------------------------
class DeepNN(nn.Module):
    def __init__(self, input_size, hidden1, hidden2, num_classes):
        super(DeepNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden1)  # input → hidden1
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden1, hidden2)     # hidden1 → hidden2
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden2, num_classes) # hidden2 → output

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        return out

# ------------------------------
# 3. Initialize Model
# ------------------------------
input_size = X.shape[1]   # 4 features
hidden1, hidden2 = 16, 8
num_classes = 3          # iris has 3 classes
model = DeepNN(input_size, hidden1, hidden2, num_classes)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
# ------------------------------
# 4. Training
# ------------------------------
epochs = 100
for epoch in range(epochs):
    # Forward pass
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

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

    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

# ------------------------------
# 5. Evaluation
# ------------------------------
with torch.no_grad():
    test_outputs = model(X_test)
    _, predicted = torch.max(test_outputs, 1)
    accuracy = (predicted == y_test).sum().item() / y_test.size(0)
print("\nClassification Accuracy on Test Data: {:.2f}%".format(accuracy * 100))
```

Output:

```
Epoch [10/100], Loss: 0.8479
Epoch [20/100], Loss: 0.4921
Epoch [30/100], Loss: 0.3328
Epoch [40/100], Loss: 0.2353
Epoch [50/100], Loss: 0.1369
Epoch [60/100], Loss: 0.0789
Epoch [70/100], Loss: 0.0570
Epoch [80/100], Loss: 0.0504
Epoch [90/100], Loss: 0.0473
Epoch [100/100], Loss: 0.0458

Classification Accuracy on Test Data: 100.00%
```

Program 3

Design and implement a Convolutional Neural Network (CNN) for classification of image dataset.

Install Required Packages

```
pip install torch torchvision
```

Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms

# ------------------------------
# 1. Load Dataset (MNIST)
# ------------------------------
transform = transforms.Compose([transforms.ToTensor()])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
                                           download=True, transform=transform)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
                                          download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=64, shuffle=False)

# ------------------------------
# 2. Define CNN Model
# ------------------------------
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)   # 1→16
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)  # 16→32
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)  # 10 classes (digits 0-9)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))   # [1,28,28] → [16,14,14]
        x = self.pool(F.relu(self.conv2(x)))   # [16,14,14] → [32,7,7]
        x = x.view(-1, 32 * 7 * 7)             # flatten
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# ------------------------------
# 3. Initialize Model
# ------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ------------------------------
# 4. Training
# ------------------------------
epochs = 5
for epoch in range(epochs):
    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}/{epochs}, Loss: {running_loss/len(train_loader):.4f}")


# ------------------------------
# 5. Evaluation
# ------------------------------
correct, total = 0, 0
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()

print(f"\nTest Accuracy: {100 * correct / total:.2f}%")
```

Output:

```
100%|██████████████████████████████████████████████████████████████████████
█| 9.91M/9.91M [01:10<00:00, 141kB/s]
100%|██████████████████████████████████████████████████████████████████████
| 28.9k/28.9k [00:02<00:00, 13.7kB/s]
100%|██████████████████████████████████████████████████████████████████████
█| 1.65M/1.65M [00:15<00:00, 107kB/s]
100%|██████████████████████████████████████████████████████████████████████
████████| 4.54k/4.54k [00:00<?, ?B/s]
Epoch 1/5, Loss: 0.2158
Epoch 2/5, Loss: 0.0617
Epoch 3/5, Loss: 0.0438
Epoch 4/5, Loss: 0.0333
Epoch 5/5, Loss: 0.0269

Test Accuracy: 99.00%
```

Program 4

Build and demonstrate an auto encoder network using neural layers for data compression on image dataset.

Install Required Packages

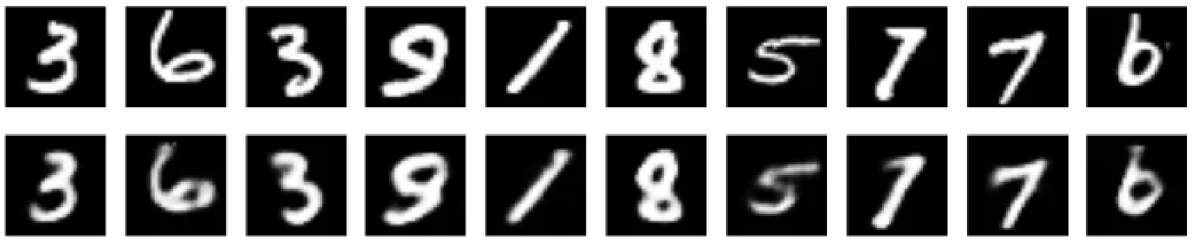
```
pip install torch torchvision matplotlib
```

Program:

```
import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt


# ------------------------------

# 1. Load Dataset (MNIST)

# ------------------------------

transform = transforms.Compose([transforms.ToTensor()])


train_dataset = torchvision.datasets.MNIST(root='./data', train=True,

                                           download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,

                                           batch_size=128, shuffle=True)


# ------------------------------

# 2. Define Autoencoder

# ------------------------------

class Autoencoder(nn.Module):

    def __init__(self):

        super(Autoencoder, self).__init__()

        # Encoder (compression)

        self.encoder = nn.Sequential(

            nn.Linear(28 * 28, 128),

            nn.ReLU(True),

            nn.Linear(128, 64),

            nn.ReLU(True),
            nn.Linear(64, 32)   # Compressed representation
        )
        # Decoder (reconstruction)
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 28 * 28),
            nn.Sigmoid()   # output values between 0-1
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# ------------------------------
# 3. Initialize Model
# ------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ------------------------------
# 4. Training
# ------------------------------
epochs = 10
for epoch in range(epochs):
    running_loss = 0.0
    for images, _ in train_loader:
        images = images.view(-1, 28*28).to(device)  # Flatten
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, images)  # Compare with original
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader):.4f}")

# ------------------------------
# 5. Demonstrate Compression & Reconstruction
# ------------------------------
# Get some test images
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
                                          download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=10, shuffle=True)

# Show original and reconstructed
model.eval()
with torch.no_grad():
    images, _ = next(iter(test_loader))
    images = images.view(-1, 28*28).to(device)
    outputs = model(images)

    # Move to CPU and reshape
    images = images.view(-1, 1, 28, 28).cpu()
    outputs = outputs.view(-1, 1, 28, 28).cpu()

    # Plot
    fig, axes = plt.subplots(2, 10, figsize=(10, 2))
    for i in range(10):
        # Original
        axes[0][i].imshow(images[i].squeeze(), cmap='gray')
        axes[0][i].axis("off")
        # Reconstructed
        axes[1][i].imshow(outputs[i].squeeze(), cmap='gray')
        axes[1][i].axis("off")
    plt.show()
```

Output:

```
Epoch 1/10, Loss: 0.0603
Epoch 2/10, Loss: 0.0309
Epoch 3/10, Loss: 0.0249
Epoch 4/10, Loss: 0.0218
Epoch 5/10, Loss: 0.0196
Epoch 6/10, Loss: 0.0176
Epoch 7/10, Loss: 0.0162
Epoch 8/10, Loss: 0.0150
Epoch 9/10, Loss: 0.0141
Epoch 10/10, Loss: 0.0132
```



Program 5

Design and implement a deep learning network for classification of textual documents.

Install Required Packages

```
pip install tensorflow scikit-learn numpy
```

Program:

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

print("Step 1: Loading dataset ...")
data = fetch_20newsgroups(subset='all')
texts, labels = data.data, data.target
num_classes = len(set(labels))
print(f"Loaded {len(texts)} documents across {num_classes} classes.")

print("\nStep 2: Preprocessing text ...")
```

```
MAX_WORDS = 10000   # vocabulary size
MAX_LEN = 300       # max tokens per document

tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)

X = tokenizer.texts_to_sequences(texts)
X = pad_sequences(X, maxlen=MAX_LEN)
y = to_categorical(labels, num_classes=num_classes)
print("Text converted to padded sequences.")

print("\nStep 3: Splitting into train and test ...")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print(f"Training samples: {len(X_train)}, Testing samples: {len(X_test)}")

print("\nStep 4: Building deep learning model ...")
model = Sequential([
    Embedding(input_dim=MAX_WORDS, output_dim=128, input_length=MAX_LEN),
    LSTM(128),
    Dense(num_classes, activation="softmax")
])
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
print(model.summary())

print("\nStep 5: Training model ...")
history = model.fit(X_train, y_train,
          epochs=5,
          batch_size=64,
          validation_split=0.1,
          verbose=2)

print("\nStep 6: Evaluating model on test set ...")
```

```
loss, acc = model.evaluate(X_test, y_test, verbose=0)

print(f"Test Accuracy: {acc:.4f}")
```

Output:

```
Step 1: Loading dataset ...
Loaded 18846 documents across 20 classes.

Step 2: Preprocessing text ...
Text converted to padded sequences.

Step 3: Splitting into train and test ...
Training samples: 15076, Testing samples: 3770

Step 4: Building deep learning model ...
C:\Users\Dharani\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:97
: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃        Par
am # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━
━━━━━┩
│ embedding (Embedding)                 │ ?                            │     0 (unbu
ilt) │
├───────────────────────────────────────┼──────────────────────────────┼────────────
─────┤
│ lstm (LSTM)                           │ ?                            │     0 (unbu
ilt) │
├───────────────────────────────────────┼──────────────────────────────┼────────────
─────┤
│ dense (Dense)                         │ ?                            │     0 (unbu
ilt) │
└───────────────────────────────────────┴──────────────────────────────┴────────────
─────┘

 Total params: 0 (0.00 B)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 0 (0.00 B)

None

Step 5: Training model ...
Epoch 1/5
212/212 - 146s - 688ms/step - accuracy: 0.1622 - loss: 2.7095 - val_accuracy: 0.24
87 - val_loss: 2.5639
Epoch 2/5
212/212 - 149s - 702ms/step - accuracy: 0.3419 - loss: 2.2098 - val_accuracy: 0.30
50 - val_loss: 2.3542
Epoch 3/5
```

```
212/212 - 147s - 693ms/step - accuracy: 0.4573 - loss: 1.6720 - val_accuracy: 0.44
30 - val_loss: 1.6905
Epoch 4/5
212/212 - 181s - 855ms/step - accuracy: 0.5579 - loss: 1.3637 - val_accuracy: 0.53
32 - val_loss: 1.4359
Epoch 5/5
212/212 - 115s - 545ms/step - accuracy: 0.6802 - loss: 0.9718 - val_accuracy: 0.59
68 - val_loss: 1.2279

Step 6: Evaluating model on test set ...
Test Accuracy: 0.6027
```

Program 6

Design and implement a deep learning network for forecasting time series data

Install Required Packages

```
pip install numpy scikit-learn tensorflow
```

Program:

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, TimeDistributed

# ---------------------------
# 1) Generate synthetic time series
def generate_series(n=1000):
    t = np.arange(n)
    trend = 0.01 * t
    seasonal = np.sin(2 * np.pi * t / 50)
    noise = 0.1 * np.random.randn(n)
    series = 10 + trend + seasonal + noise
    return series

series = generate_series(1000)

# ---------------------------
# 2) Scale data
```

```
scaler = StandardScaler()
series_scaled = scaler.fit_transform(series.reshape(-1,1)).flatten()

# ---------------------------
# 3) Windowing
INPUT_WINDOW = 30
OUTPUT_WINDOW = 10

def make_windows(data, input_w, output_w):
    X, Y = [], []
    for i in range(len(data) - input_w - output_w):
        X.append(data[i:i+input_w])
        Y.append(data[i+input_w:i+input_w+output_w])
    return np.array(X), np.array(Y)

X, Y = make_windows(series_scaled, INPUT_WINDOW, OUTPUT_WINDOW)
X = X[..., np.newaxis]
Y = Y[..., np.newaxis]

# Train/test split (80/20)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]

# ---------------------------
# 4) Build Seq2Seq LSTM model
# Encoder
encoder_inputs = Input(shape=(INPUT_WINDOW,1))
encoder_lstm = LSTM(64, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_inputs)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(OUTPUT_WINDOW,1))
decoder_lstm = LSTM(64, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
```

```
decoder_dense = TimeDistributed(Dense(1))
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Prepare decoder input (teacher forcing)
decoder_input_train = np.zeros_like(Y_train)
decoder_input_test = np.zeros_like(Y_test)

# ---------------------------
# 5) Train
history = model.fit([X_train, decoder_input_train], Y_train,
                    validation_data=([X_test, decoder_input_test], Y_test),
                    epochs=20, batch_size=32, verbose=1)

# ---------------------------
# 6) Evaluate
preds = model.predict([X_test, decoder_input_test])
preds_unscaled = scaler.inverse_transform(preds.reshape(-1,1)).reshape(preds.shape)
Y_test_unscaled = scaler.inverse_transform(Y_test.reshape(-1,1)).reshape(Y_test.shape)

mae = mean_absolute_error(Y_test_unscaled.flatten(), preds_unscaled.flatten())
rmse = np.sqrt(mean_squared_error(Y_test_unscaled.flatten(), preds_unscaled.flatten()))
print(f"Test MAE: {mae:.4f}, Test RMSE: {rmse:.4f}")
```

Output:

```
Epoch 1/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 10s 106ms/step - loss: 0.1861 - mae: 0.3389 - val_loss:
0.0869 - val_mae: 0.2378
Epoch 2/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0688 - mae: 0.2244 - val_loss: 0
.1616 - val_mae: 0.3316
Epoch 3/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - loss: 0.0576 - mae: 0.2026 - val_loss: 0
.0898 - val_mae: 0.2380
Epoch 4/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 0.0445 - mae: 0.1761 - val_loss: 0
.0656 - val_mae: 0.2053
Epoch 5/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 0.0276 - mae: 0.1327 - val_loss: 0
.0478 - val_mae: 0.1746
```

```
Epoch 6/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - loss: 0.0085 - mae: 0.0700 - val_loss: 0
.0473 - val_mae: 0.1834
Epoch 7/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0043 - mae: 0.0514 - val_loss: 0
.0455 - val_mae: 0.1831
Epoch 8/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - loss: 0.0036 - mae: 0.0474 - val_loss: 0
.0383 - val_mae: 0.1701
Epoch 9/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - loss: 0.0034 - mae: 0.0460 - val_loss: 0
.0367 - val_mae: 0.1668
Epoch 10/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 0.0030 - mae: 0.0435 - val_loss: 0
.0264 - val_mae: 0.1397
Epoch 11/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - loss: 0.0029 - mae: 0.0423 - val_loss: 0
.0252 - val_mae: 0.1366
Epoch 12/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0027 - mae: 0.0414 - val_loss: 0
.0198 - val_mae: 0.1188
Epoch 13/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0028 - mae: 0.0417 - val_loss: 0
.0242 - val_mae: 0.1337
Epoch 14/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0027 - mae: 0.0412 - val_loss: 0
.0320 - val_mae: 0.1547
Epoch 15/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 0.0029 - mae: 0.0422 - val_loss: 0
.0214 - val_mae: 0.1227
Epoch 16/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - loss: 0.0027 - mae: 0.0407 - val_loss: 0
.0214 - val_mae: 0.1246
Epoch 17/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0025 - mae: 0.0394 - val_loss: 0
.0126 - val_mae: 0.0898
Epoch 18/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0026 - mae: 0.0404 - val_loss: 0
.0091 - val_mae: 0.0737
Epoch 19/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0026 - mae: 0.0403 - val_loss: 0
.0127 - val_mae: 0.0885
Epoch 20/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0026 - mae: 0.0399 - val_loss: 0
.0100 - val_mae: 0.0787
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step
Test MAE: 0.2315, Test RMSE: 0.2943
```

Program 7

Write a program to enable pre-train models to classify a given image dataset.

```
project_folder/          ← Your main project folder
 ├── main.py               ← Python script
 └── data/               ← Dataset folder
     ├── train/          ← Training images
     │    ├── cat/       ← Class 1 training images
     │    │    ├── cat1.jpg
     │    │    ├── cat2.jpg
     │    │    └── ...
     │    └── dog/        ← Class 2 training images
     │        ├── dog1.jpg
     │        ├── dog2.jpg
     │        └── ...
     └── val/            ← Validation images
         ├── cat/        ← Class 1 validation images
         │    ├── cat3.jpg
         │    └── ...
         └── dog/         ← Class 2 validation images
             ├── dog3.jpg
             └── ...
```

Install Required Packages

```
pip install tensorflow
```

Program:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

# 1) Dataset paths adjust as per requirement
train_dir = "data/train"
val_dir = "data/val"
img_size = (224, 224)
```

```
batch_size = 32

# 2) Data generators
train_gen = ImageDataGenerator(preprocessing_function=preprocess_input).flow_from_directory(
    train_dir, target_size=img_size, batch_size=batch_size, class_mode='categorical'
)
val_gen = ImageDataGenerator(preprocessing_function=preprocess_input).flow_from_directory(
    val_dir, target_size=img_size, batch_size=batch_size, class_mode='categorical'
)

num_classes = train_gen.num_classes

# 3) Pre-trained model (ResNet50) without top layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))
base_model.trainable = False  # Freeze base model

# 4) Add classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# 5) Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 6) Train model
model.fit(train_gen, validation_data=val_gen, epochs=5)

# 7) Evaluate model
loss, acc = model.evaluate(val_gen)
print(f"Validation Loss: {loss:.4f}, Validation Accuracy: {acc:.4f}")
```

Output:

```
Found 4 images belonging to 3 classes.
Found 4 images belonging to 3 classes.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 14s 14s/step - accuracy: 0.2500 - loss: 2.7132 - val_accuracy: 0.7500 - val_loss: 0.4020
Epoch 2/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 1.0000 - loss: 0.1400 - val_accuracy: 1.0000 - val_loss: 0.2610
Epoch 3/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 1.0000 - loss: 0.0120 - val_accuracy: 1.0000 - val_loss: 0.1789
Epoch 4/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 1.0000 - val_loss: 0.1277
Epoch 5/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 1.0000 - loss: 6.7348e-04 - val_accuracy: 1.0000 - val_loss: 0.0973
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 1.0000 - loss: 0.0973
Validation Loss: 0.0973, Validation Accuracy: 1.0000
```

Program 8

Simple Grid World Problem: Design a custom 2D grid world where the agent navigates from a start position to a goal, avoiding obstacles. Environment: Custom grid (easily implemented in Python).

Program:

```
import random
class GridWorld:
    def __init__(self, width, height, start, goal, obstacles=[]):
        # Initialize grid dimensions, start & goal positions, and obstacles
        self.width = width
        self.height = height
        self.start = start
        self.goal = goal
        self.obstacles = obstacles
        self.reset()

    def reset(self):
        # Reset the agent to the start position
        self.agent_pos = self.start
        return self.agent_pos

    def render(self):
        # Print the current grid with agent, start, goal, and obstacles
        for i in range(self.height):
            row = ''
            for j in range(self.width):
                if (i, j) == self.agent_pos:
                    row += 'A '  # Agent's current position
                elif (i, j) == self.start:
                    row += 'S '  # Start position
                elif (i, j) == self.goal:
                    row += 'G '  # Goal position
                elif (i, j) in self.obstacles:
                    row += 'O '  # Obstacle
                else:
                    row += '. '  # Empty cell
            print(row)
        print()

    def step(self, action):
        # Move the agent in the given direction: 'up', 'down', 'left', 'right'
        x, y = self.agent_pos
        if action == 'up':
            x -= 1
        elif action == 'down':
            x += 1
        elif action == 'left':
            y -= 1
        elif action == 'right':
            y += 1
        else:
            print("Invalid action! Use: up, down, left, right")

        # Check if new position is within grid boundaries
        if 0 <= x < self.height and 0 <= y < self.width:
            new_pos = (x, y)
        else:
            new_pos = self.agent_pos  # Stay in place if moving out of bounds

        self.agent_pos = new_pos

        # Determine reward and whether the episode is done
        if new_pos == self.goal:
            reward = 1  # Reached goal
            done = True
        elif new_pos in self.obstacles:
            reward = -1  # Hit an obstacle
            done = True
        else:
            reward = -0.01  # Small penalty for each step
            done = False
        return new_pos, reward, done



# -----------------------------

# User-controlled agent with RANDOM obstacles

# -----------------------------

width, height = 5, 5

start = (0, 0)

goal = (4, 4)


# Generate random obstacles (3 positions each run)

all_positions = [(i, j) for i in range(height) for j in range(width)]

all_positions.remove(start)

all_positions.remove(goal)

obstacles = random.sample(all_positions, 3)


env = GridWorld(width, height, start, goal, obstacles)

state = env.reset()

done = False


print("Welcome to Grid World!\n")

print("Game Rules:")

print(" - Reach the Goal (G) to win 🎉")

print(" - Avoid Obstacles (O), or you lose 💀")

print(" - You are the Agent (A), starting from Start (S)")

print(" - Use commands: up, down, left, right to move\n")


env.render()


while not done:

    action = input("Enter your move (up, down, left, right): ")

    state, reward, done = env.step(action)

    print(f"New State: {state}, Reward: {reward}")

    env.render()


if state == goal:
    print("🎉 Congratulations! You reached the goal!")
else:
    print("💀 Game Over! You hit an obstacle.")
```

Output:

```
Welcome to Grid World!

Game Rules:
 - Reach the Goal (G) to win 🎉
 - Avoid Obstacles (O), or you lose 💀
 - You are the Agent (A), starting from Start (S)
 - Use commands: up, down, left, right to move

A . . . O
. . . . O
. . . . .
. O . . .
. . . . G

Enter your move (up, down, left, right):  right
New State: (0, 1), Reward: -0.01
S A . . O
. . . . O
. . . . .
. O . . .
. . . . G

Enter your move (up, down, left, right):  down
New State: (1, 1), Reward: -0.01
S . . . O
. A . . O
. . . . .
. O . . .
. . . . G

Enter your move (up, down, left, right):  down
New State: (2, 1), Reward: -0.01
S . . . O
. . . . O
. A . . .
. O . . .
. . . . G

Enter your move (up, down, left, right):  down
New State: (3, 1), Reward: -1
S . . . O
. . . . O
. . . . .
. A . . .
. . . . G

💀 Game Over! You hit an obstacle.
```