

7CAI4-21: Deep Learning and its Applications Lab

B.Tech. Computer Science and Engineering (AI) - VII Semester

Python Programs for Experiments 1 to 4

SN	Experiment	Aim
1	Experiment 1(a)	Linear regression with a single variable using a deep neural network
2	Experiment 1(b)	Linear regression with multiple variables using a deep neural network
3	Experiment 2(a)	Speech into text
4	Experiment 2(b)	Text into speech
5	Experiment 2(c)	Video into frames
6	Experiment 3	Feed forward neural network for logic gates
7	Experiment 4(a)	Character recognition using CNN
8	Experiment 4(b)	Character recognition using RNN

Setup: install Python 3.9+ and run `pip install tensorflow numpy matplotlib scikit-learn SpeechRecognition pyaudio gTTS pyttsx3 opencv-python`. Each program is self-contained and can be run with `python filename.py`.

Experiment 1(a): Linear regression with a single variable using a deep neural network

Aim

To build a deep neural network model starting from linear regression with a single input variable.

Approach

A single Dense neuron without activation is exactly linear regression ($y = wx + b$). Hidden ReLU layers are then added to form a deep network. Both models are trained with the Adam optimizer on the MSE loss and compared on synthetic data $y = 3x + 2 + \text{noise}$.

Libraries

tensorflow, numpy, matplotlib, scikit-learn

Program (exp1a_linear_regression_single_variable.py)

```
"""
Experiment 1(a): Deep neural network starting with linear regression
(single input variable)

Requirements: pip install tensorflow numpy matplotlib scikit-learn
"""
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split

np.random.seed(42)
tf.random.set_seed(42)

# -----
# 1. Create a synthetic dataset:  $y = 3x + 2 + \text{noise}$ 
# -----
X = np.random.uniform(-10, 10, size=(500, 1)).astype("float32")
y = (3 * X + 2 + np.random.normal(0, 1.5, size=(500, 1))).astype("float32")

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

# -----
# 2. Step 1: plain linear regression = ONE neuron, no activation
# -----
linear_model = keras.Sequential([
    keras.Input(shape=(1,)),
    layers.Dense(1)  #  $y = w \cdot x + b$ 
])
linear_model.compile(optimizer=keras.optimizers.Adam(0.1), loss="mse")
linear_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

w, b = linear_model.layers[0].get_weights()
print(f"Linear model learned :  $y = \{w[0][0]:.3f\} \cdot x + \{b[0]:.3f\}$  (true:  $3x + 2$ )")
print("Linear model test MSE:", linear_model.evaluate(X_test, y_test, verbose=0))

# -----
# 3. Step 2: extend it into a DEEP neural network (hidden layers)
# -----
deep_model = keras.Sequential([
    keras.Input(shape=(1,)),
    layers.Dense(32, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1)  # linear output for regression
])
deep_model.compile(optimizer=keras.optimizers.Adam(0.01), loss="mse",
```

```

        metrics=["mae"])
deep_model.summary()

history = deep_model.fit(X_train, y_train, validation_split=0.2,
                        epochs=100, batch_size=32, verbose=0)

loss, mae = deep_model.evaluate(X_test, y_test, verbose=0)
print(f"Deep model test MSE: {loss:.4f}    MAE: {mae:.4f}")

# -----
# 4. Plot results
# -----
x_line = np.linspace(-10, 10, 200).reshape(-1, 1).astype("float32")

plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_test, y_test, s=10, label="Test data")
plt.plot(x_line, linear_model.predict(x_line, verbose=0), "r", label="Linear (1 neuron)")
plt.plot(x_line, deep_model.predict(x_line, verbose=0), "g--", label="Deep NN")
plt.xlabel("x"); plt.ylabel("y"); plt.legend(); plt.title("Single-variable regression")

plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"], label="val loss")
plt.xlabel("Epoch"); plt.ylabel("MSE"); plt.legend(); plt.title("Training curve")
plt.tight_layout()
plt.savefig("exp1a_result.png", dpi=120)
plt.show()

# Predict for a new value
new_x = np.array([[4.0]], dtype="float32")
print("Prediction for x = 4 :", deep_model.predict(new_x, verbose=0)[0][0], "(expected ~14)")

```

Experiment 1(b): Linear regression with multiple variables using a deep neural network

Aim

To build a deep neural network model starting from linear regression with multiple input variables.

Approach

Three input features are standardised and passed first to one linear neuron, then to a deep network of three hidden layers. Performance is reported using MSE, MAE and R2 score.

Libraries

tensorflow, numpy, matplotlib, scikit-learn

Program (exp1b_linear_regression_multiple_variables.py)

```
"""
Experiment 1(b): Deep neural network starting with linear regression
(multiple input variables)

Requirements: pip install tensorflow numpy matplotlib scikit-learn
"""
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

np.random.seed(42)
tf.random.set_seed(42)

# -----
# 1. Synthetic dataset with 3 input variables
#    $y = 4x_1 - 2x_2 + 0.5x_3 + 7 + \text{noise}$ 
# -----
n = 1000
X = np.random.uniform(-5, 5, size=(n, 3)).astype("float32")
true_w = np.array([4.0, -2.0, 0.5], dtype="float32")
y = (X @ true_w + 7 + np.random.normal(0, 1.0, size=n)).astype("float32")
y = y.reshape(-1, 1)

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

# Feature scaling helps gradient descent converge
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# -----
# 2. Step 1: linear regression with multiple variables (1 neuron)
# -----
linear_model = keras.Sequential([
    keras.Input(shape=(3,)),
    layers.Dense(1)
])
#  $y = w_1x_1 + w_2x_2 + w_3x_3 + b$ 
linear_model.compile(optimizer=keras.optimizers.Adam(0.05), loss="mse")
linear_model.fit(X_train_s, y_train, epochs=100, batch_size=32, verbose=0)

pred_lin = linear_model.predict(X_test_s, verbose=0)
print("Linear model R2 :", round(r2_score(y_test, pred_lin), 4))

# -----
# 3. Step 2: deep neural network
```

```

# -----
deep_model = keras.Sequential([
    keras.Input(shape=(3,)),
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1)
])
deep_model.compile(optimizer=keras.optimizers.Adam(0.005), loss="mse",
                  metrics=["mae"])
deep_model.summary()

history = deep_model.fit(X_train_s, y_train, validation_split=0.2,
                        epochs=100, batch_size=32, verbose=0)

loss, mae = deep_model.evaluate(X_test_s, y_test, verbose=0)
pred_deep = deep_model.predict(X_test_s, verbose=0)
print(f"Deep model    MSE: {loss:.4f}    MAE: {mae:.4f}    R2: {r2_score(y_test, pred_deep):.4f}")

# -----
# 4. Plots
# -----
plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"], label="val loss")
plt.xlabel("Epoch"); plt.ylabel("MSE"); plt.legend(); plt.title("Training curve")

plt.subplot(1, 2, 2)
plt.scatter(y_test, pred_deep, s=10)
lims = [y_test.min(), y_test.max()]
plt.plot(lims, lims, "r--")
plt.xlabel("Actual y"); plt.ylabel("Predicted y"); plt.title("Actual vs predicted")
plt.tight_layout()
plt.savefig("exp1b_result.png", dpi=120)
plt.show()

# Predict for a new sample
sample = np.array([[1.0, 2.0, 3.0]], dtype="float32")
print("Prediction for [1, 2, 3]:",
      deep_model.predict(scaler.transform(sample), verbose=0)[0][0],
      "(expected ~", float(sample @ true_w + 7), ")")

```

Experiment 2(a): Speech into text

Aim

To write a program that converts speech into text.

Approach

The SpeechRecognition library captures audio from a WAV file or the microphone and sends it to the Google Web Speech API, which returns the transcribed text.

Libraries

SpeechRecognition, pyaudio (internet required)

Program (exp2a_speech_to_text.py)

```
"""
Experiment 2(a): Convert speech into text

Two modes are provided:
1. From an audio file (WAV/AIFF/FLAC)
2. Live from the microphone

Requirements:
    pip install SpeechRecognition pyaudio
Note: recognize_google() needs an internet connection.
On Windows, if pyaudio fails to install: pip install pipwin && pipwin install pyaudio
"""
import speech_recognition as sr

recognizer = sr.Recognizer()

def speech_from_file(path):
    """Convert a WAV/AIFF/FLAC audio file to text."""
    with sr.AudioFile(path) as source:
        recognizer.adjust_for_ambient_noise(source, duration=0.5)
        audio = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio)
        print("Recognized text:", text)
        return text
    except sr.UnknownValueError:
        print("Could not understand the audio.")
    except sr.RequestError as e:
        print("Speech service error:", e)

def speech_from_microphone():
    """Listen on the microphone and convert speech to text."""
    with sr.Microphone() as source:
        print("Adjusting for background noise...")
        recognizer.adjust_for_ambient_noise(source, duration=1)
        print("Speak now...")
        audio = recognizer.listen(source, timeout=10, phrase_time_limit=15)
    try:
        text = recognizer.recognize_google(audio)
        print("You said:", text)
        return text
    except sr.UnknownValueError:
        print("Could not understand the audio.")
    except sr.RequestError as e:
        print("Speech service error:", e)

if __name__ == "__main__":
    choice = input("Enter 1 for audio file, 2 for microphone: ").strip()
    if choice == "1":
        speech_from_file(input("Path to audio file (.wav): ").strip())
    else:
```

`speech_from_microphone()`

Experiment 2(b): Text into speech

Aim

To write a program that converts text into speech.

Approach

gTTS generates an MP3 file online; pyttsx3 is an offline engine that speaks the text directly and allows control of the speaking rate, volume and voice.

Libraries

gTTS, pyttsx3

Program (exp2b_text_to_speech.py)

```
"""
Experiment 2(b): Convert text into speech

Two approaches:
1. gTTS - Google Text-to-Speech (online, saves an MP3 file)
2. pyttsx3 - offline engine that speaks directly

Requirements:
    pip install gTTS pyttsx3
"""
from gtts import gTTS
import pyttsx3
import os

def text_to_speech_gtts(text, filename="output_gtts.mp3", lang="en"):
    """Online conversion; saves the speech as an MP3 file."""
    tts = gTTS(text=text, lang=lang, slow=False)
    tts.save(filename)
    print(f"Speech saved to {filename}")
    # Play the file (works on Windows; use 'afplay' on macOS / 'mpg123' on Linux)
    if os.name == "nt":
        os.system(f'start {filename}')
```

```
def text_to_speech_offline(text, rate=150, volume=1.0):
    """Offline conversion; speaks the text aloud."""
    engine = pyttsx3.init()
    engine.setProperty("rate", rate)          # words per minute
    engine.setProperty("volume", volume)      # 0.0 to 1.0
    voices = engine.getProperty("voices")
    if voices:
        engine.setProperty("voice", voices[0].id)
    engine.say(text)
    engine.runAndWait()
```

```
if __name__ == "__main__":
    text = input("Enter the text to convert into speech: ").strip()
    if not text:
        text = "Hello! This is a deep learning lab experiment on text to speech."

    print("1. gTTS (online, saves MP3)")
    print("2. pyttsx3 (offline, speaks directly)")
    choice = input("Choose an option: ").strip()

    if choice == "1":
        text_to_speech_gtts(text)
    else:
        text_to_speech_offline(text)
```

Experiment 2(c): Video into frames

Aim

To write a program that converts a video into individual frames.

Approach

OpenCV reads the video frame by frame with VideoCapture and saves every n-th frame as a JPEG image in an output folder, after printing the video's FPS, frame count and resolution.

Libraries

opencv-python

Program (exp2c_video_to_frames.py)

```
"""
Experiment 2(c): Convert a video into frames

Requirements:
    pip install opencv-python
"""
import cv2
import os

def video_to_frames(video_path, output_dir="frames", every_n=1, max_frames=None):
    """
    Read a video and save its frames as JPEG images.

    video_path : path of the input video
    output_dir  : folder in which frames are stored
    every_n     : save every n-th frame (1 = all frames)
    max_frames  : optional limit on number of saved frames
    """
    if not os.path.exists(video_path):
        raise FileNotFoundError(f"Video not found: {video_path}")

    os.makedirs(output_dir, exist_ok=True)
    cap = cv2.VideoCapture(video_path)

    fps = cap.get(cv2.CAP_PROP_FPS)
    total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    print(f"Video info -> FPS: {fps:.2f}, Frames: {total}, Size: {width}x{height}")

    count, saved = 0, 0
    while True:
        success, frame = cap.read()
        if not success:
            break  # end of video

        if count % every_n == 0:
            filename = os.path.join(output_dir, f"frame_{saved:05d}.jpg")
            cv2.imwrite(filename, frame)
            saved += 1
            if max_frames and saved >= max_frames:
                break
        count += 1

    cap.release()
    print(f"Done. Read {count} frames, saved {saved} frames in '{output_dir}/'")
    return saved

if __name__ == "__main__":
    path = input("Enter video file path (e.g. sample.mp4): ").strip()
    step = input("Save every n-th frame (default 1): ").strip()
    step = int(step) if step.isdigit() and int(step) > 0 else 1
```

```
video_to_frames(path, output_dir="frames", every_n=step)
```

Experiment 3: Feed forward neural network for logic gates

Aim

To build a feed forward neural network for the prediction of logic gates.

Approach

A 2-8-1 network (tanh hidden layer, sigmoid output) is trained separately for AND, OR, NAND, NOR, XOR and XNOR. XOR and XNOR are not linearly separable, which is why a hidden layer is required.

Libraries

tensorflow, numpy

Program (exp3_logic_gates_feedforward.py)

```
"""
Experiment 3: Feed forward neural network for prediction of logic gates
(AND, OR, NAND, NOR, XOR, XNOR)

A single neuron can learn the linearly separable gates (AND, OR, NAND, NOR).
XOR / XNOR are NOT linearly separable, so they need a hidden layer.

Requirements: pip install tensorflow numpy
"""
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

tf.random.set_seed(1)
np.random.seed(1)

# Inputs common to all two-input gates
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype="float32")

gates = {
    "AND": np.array([0, 0, 0, 1], dtype="float32"),
    "OR": np.array([0, 1, 1, 1], dtype="float32"),
    "NAND": np.array([1, 1, 1, 0], dtype="float32"),
    "NOR": np.array([1, 0, 0, 0], dtype="float32"),
    "XOR": np.array([0, 1, 1, 0], dtype="float32"),
    "XNOR": np.array([1, 0, 0, 1], dtype="float32"),
}

def build_model():
    """2 inputs -> 8 hidden neurons (tanh) -> 1 output (sigmoid)."""
    model = keras.Sequential([
        keras.Input(shape=(2,)),
        layers.Dense(8, activation="tanh"),
        layers.Dense(1, activation="sigmoid"),
    ])
    model.compile(optimizer=keras.optimizers.Adam(0.05),
                  loss="binary_crossentropy", metrics=["accuracy"])
    return model

trained = {}
for name, y in gates.items():
    model = build_model()
    model.fit(X, y, epochs=500, batch_size=4, verbose=0)
    loss, acc = model.evaluate(X, y, verbose=0)
    preds = model.predict(X, verbose=0).flatten()

    print(f"\n=== {name} gate === accuracy = {acc * 100:.0f}%")
    print(" A B | target | output | predicted")
    for (a, b), t, p in zip(X.astype(int), y.astype(int), preds):
        print(f" {a} {b} | {t} | {p:.3f} | {int(p > 0.5)}")
    trained[name] = model
```

```

# -----
# Interactive test
# -----
print("\nTest a gate yourself (type 'q' to quit)")
while True:
    g = input(f"Gate {list(gates)}: ").strip().upper()
    if g == "Q":
        break
    if g not in trained:
        print("Unknown gate.")
        continue
    a = int(input("Input A (0/1): "))
    b = int(input("Input B (0/1): "))
    out = trained[g].predict(np.array([[a, b]], dtype="float32"), verbose=0)[0][0]
    print(f"{g}({a}, {b}) = {int(out > 0.5)} (raw output {out:.3f})")

```

Experiment 4(a): Character recognition using CNN

Aim

To write a program for character (handwritten digit) recognition using a Convolutional Neural Network.

Approach

Two Conv2D + MaxPooling blocks extract spatial features from the 28x28 MNIST images, followed by a dropout and dense classifier with a 10-way softmax output.

Libraries

tensorflow, numpy, matplotlib, scikit-learn

Program (exp4a_character_recognition_cnn.py)

```
"""
Experiment 4(a): Character (handwritten digit) recognition using CNN
Dataset: MNIST (60,000 train + 10,000 test images of 28x28 digits)

Requirements: pip install tensorflow numpy matplotlib scikit-learn
"""
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import classification_report, confusion_matrix

tf.random.set_seed(42)
np.random.seed(42)

# -----
# 1. Load and pre-process data
# -----
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_train = x_train.astype("float32") / 255.0          # scale to [0, 1]
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)                # (N, 28, 28, 1)
x_test = np.expand_dims(x_test, -1)

print("Train:", x_train.shape, " Test:", x_test.shape)

# -----
# 2. Build the CNN
# -----
model = keras.Sequential([
    keras.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, (3, 3), activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax"),
])
model.summary()

model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

# -----
# 3. Train
# -----
history = model.fit(x_train, y_train, epochs=8, batch_size=128,
                    validation_split=0.1, verbose=2)
# -----
```

```

# 4. Evaluate
# -----
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nCNN Test accuracy: {test_acc * 100:.2f}%")

y_pred = np.argmax(model.predict(x_test, verbose=0), axis=1)
print(classification_report(y_test, y_pred, digits=3))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))

# -----
# 5. Plots: training curves and sample predictions
# -----
plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="train")
plt.plot(history.history["val_accuracy"], label="validation")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("CNN accuracy")
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="validation")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("CNN loss")
plt.tight_layout()
plt.savefig("exp4a_curves.png", dpi=120)

plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[i].squeeze(), cmap="gray")
    plt.title(f"True {y_test[i]} / Pred {y_pred[i]}", fontsize=8)
    plt.axis("off")
plt.tight_layout()
plt.savefig("exp4a_predictions.png", dpi=120)
plt.show()

model.save("cnn_mnist.keras")
print("Model saved as cnn_mnist.keras")

```

Experiment 4(b): Character recognition using RNN

Aim

To write a program for character (handwritten digit) recognition using a Recurrent Neural Network.

Approach

Each MNIST image is read as a sequence of 28 rows (28 pixels each) by two stacked LSTM layers; the final hidden state feeds a dense softmax classifier.

Libraries

tensorflow, numpy, matplotlib, scikit-learn

Program (exp4b_character_recognition_rnn.py)

```
"""
Experiment 4(b): Character (handwritten digit) recognition using RNN
Dataset: MNIST. Each 28x28 image is treated as a sequence of 28 time
steps, where every time step is one row of 28 pixels. An LSTM reads the
rows top-to-bottom and the last hidden state is used for classification.

Requirements: pip install tensorflow numpy matplotlib scikit-learn
"""

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import classification_report

tf.random.set_seed(42)
np.random.seed(42)

# -----
# 1. Load and pre-process data
# -----
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0      # shape (N, 28, 28)
x_test = x_test.astype("float32") / 255.0        # (timesteps=28, features=28)

print("Train:", x_train.shape, " Test:", x_test.shape)

# -----
# 2. Build the RNN (stacked LSTM)
# -----
model = keras.Sequential([
    keras.Input(shape=(28, 28)),
    layers.LSTM(128, return_sequences=True),
    layers.LSTM(64),
    layers.Dropout(0.3),
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax"),
])
model.summary()

model.compile(optimizer=keras.optimizers.Adam(0.002),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

# -----
# 3. Train
# -----
history = model.fit(x_train, y_train, epochs=8, batch_size=128,
                    validation_split=0.1, verbose=2)

# -----
# 4. Evaluate
# -----
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
```

```

print(f"\nRNN (LSTM) Test accuracy: {test_acc * 100:.2f}%")

y_pred = np.argmax(model.predict(x_test, verbose=0), axis=1)
print(classification_report(y_test, y_pred, digits=3))

# -----
# 5. Plots
# -----
plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="train")
plt.plot(history.history["val_accuracy"], label="validation")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("RNN accuracy")
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="validation")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("RNN loss")
plt.tight_layout()
plt.savefig("exp4b_curves.png", dpi=120)

plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[i], cmap="gray")
    plt.title(f"True {y_test[i]} / Pred {y_pred[i]}", fontsize=8)
    plt.axis("off")
plt.tight_layout()
plt.savefig("exp4b_predictions.png", dpi=120)
plt.show()

model.save("rnn_mnist.keras")
print("Model saved as rnn_mnist.keras")

```