Python for Computer Science and Data Science 2 (CSE 3652)

**Minor Assignment-5: DEEP LEARNING**

1. Explain briefly Single layer perceptron and multilayer perceptron with architecture and illustrate the loss function associate with it.

**Ans:-** Single Layer Perceptron (SLP)

--------------------------------

Definition:

A Single Layer Perceptron is the most basic form of a neural network. It consists of:

- An input layer

- An output layer with no hidden layers

Architecture:

Input Layer Output

(x1) ─┐

(x2) ─┼──► (weighted sum + activation) ► ŷ

(x3) ─┘

Each input is multiplied by a weight, summed with a bias, and passed through an activation function (usually step or sigmoid for binary classification).

Loss Function:

For binary classification:

L\_binary = -[y \* log(ŷ) + (1 - y) \* log(1 - ŷ)]

Where:

- y is the true label (0 or 1)

- ŷ is the predicted probability

--------------------------------------------------

Multilayer Perceptron (MLP)

------------------------------

Definition:

A Multilayer Perceptron is a feedforward neural network with one or more hidden layers between input and output. It can model non-linear relationships.

Architecture:

Input Layer Hidden Layer(s) Output Layer

(x1) ─┐ (h1) ─┐

(x2) ─┼─► W1 ─► (h2) ─┼─► W2 ─► (ŷ)

(x3) ─┘ (h3) ─┘

- Activation functions in hidden layers: ReLU, tanh, etc.

- Output layer activation: softmax (multi-class) or sigmoid (binary)

Loss Function:

For multi-class classification using softmax:

L\_categorical = -∑[i=1 to C] y\_i \* log(ŷ\_i)

Where:

- C = number of classes

- y\_i = 1 if true class is i, else 0

- ŷ\_i = predicted probability for class i

--------------------------------------------------

Summary Table:

| Aspect | SLP | MLP |

|----------------|----------------------------|--------------------------------|

| Layers | 1 (no hidden layers) | ≥2 (at least one hidden layer) |

| Function | Linear classifier | Non-linear modeling |

| Suitable for | Linearly separable data | Complex problems (e.g., images)|

| Activation | Step / Sigmoid | ReLU / Tanh / Sigmoid / Softmax |

1. How would you define the architecture of a simple feed forward ANN for classifying the Iris dataset? Write python code for the same.

**Ans:-** Definition: Feedforward ANN for Iris Classification

A simple feedforward Artificial Neural Network (ANN) for the Iris dataset has:

* Input layer: 4 neurons (sepal length, sepal width, petal length, petal width)
* Hidden layer(s): e.g., 1 layer with 8 neurons using ReLU activation
* Output layer: 3 neurons (for 3 classes: Setosa, Versicolor, Virginica) with Softmax activation

Code:

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

# Load and prepare the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target.reshape(-1, 1)

# One-hot encode labels

encoder = OneHotEncoder(sparse\_output=False)

y\_encoded = encoder.fit\_transform(y)

# Normalize input features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.2, random\_state=42)

# Build a simple feedforward ANN

model = Sequential()

model.add(Dense(8, input\_dim=4, activation='relu')) # Hidden layer

model.add(Dense(3, activation='softmax')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=5, verbose=0)

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy:.4f}')

Output: Test Accuracy: 1.0000

1. How can you build and train a simple Artificial Neural Network (ANN) using the MNIST dataset to classify handwritten digits? Write python code for this.

**Ans:-** Architecture Overview

* Input layer: 784 neurons (28×28 flattened pixels)
* Hidden layer: 128 neurons, ReLU activation
* Output layer: 10 neurons, Softmax activation (one for each digit class)

Code:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

# Load MNIST data

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# Normalize pixel values

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# One-hot encode labels

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Build ANN model

model = Sequential()

model.add(Flatten(input\_shape=(28, 28))) # Flatten 28x28 to 784

model.add(Dense(128, activation='relu')) # Hidden layer

model.add(Dense(10, activation='softmax')) # Output layer

# Compile model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics = ['accuracy'])

# Train model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, verbose=1)

# Evaluate model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy:.4f}')

Output: Test Accuracy: 0.9791

1. Find convolution , ReLu and Max Pooling with the following data Input image (4×4):

[[1, 2, 0, 1],

[3, 1, 2, 2],

[1, 0, 1, 3],

[2, 1, 2, 1]]

Filter/kernel (2×2):

[[1, 0],

[0, −1]]

**Ans:-** Input Image (4×4):

[[1, 2, 0, 1],

[3, 1, 2, 2],

[1, 0, 1, 3],

[2, 1, 2, 1]]

Filter/Kernel (2×2):

[[ 1, 0],

[ 0, -1]]

Step 1: Convolution (stride = 1, no padding)

Each 2×2 region is convolved with the filter:

(0,0): [[1,2],[3,1]] → 1×1 + 2×0 + 3×0 + 1×(-1) = 0

(0,1): [[2,0],[1,2]] → 2×1 + 0×0 + 1×0 + 2×(-1) = 0

(0,2): [[0,1],[2,2]] → 0×1 + 1×0 + 2×0 + 2×(-1) = -2

(1,0): [[3,1],[1,0]] → 3×1 + 1×0 + 1×0 + 0×(-1) = 3

(1,1): [[1,2],[0,1]] → 1×1 + 2×0 + 0×0 + 1×(-1) = 0

(1,2): [[2,2],[1,3]] → 2×1 + 2×0 + 1×0 + 3×(-1) = -1

(2,0): [[1,0],[2,1]] → 1×1 + 0×0 + 2×0 + 1×(-1) = 0

(2,1): [[0,1],[1,2]] → 0×1 + 1×0 + 1×0 + 2×(-1) = -2

(2,2): [[1,3],[2,1]] → 1×1 + 3×0 + 2×0 + 1×(-1) = 0

Convolution Output (3×3):

[[ 0, 0, -2],

[ 3, 0, -1],

[ 0, -2, 0]]

Step 2: ReLU Activation (replace negatives with 0)

[[0, 0, 0],

[3, 0, 0],

[0, 0, 0]]

Step 3: Max Pooling (2×2 window, stride=1)

Window (0,0) → [[0, 0], [3, 0]] = 3

Window (0,1) → [[0, 0], [0, 0]] = 0

Window (1,0) → [[3, 0], [0, 0]] = 3

Window (1,1) → [[0, 0], [0, 0]] = 0

Max Pooled Output (2×2):

[[3, 0],

[3, 0]]

Final Results Summary:

Convolution Output:

[[ 0, 0, -2],

[ 3, 0, -1],

[ 0, -2, 0]]

After ReLU:

[[0, 0, 0],

[3, 0, 0],

[0, 0, 0]]

After Max Pooling:

[[3, 0],

[3, 0]]

1. How can you build a Convolutional Neural Network (CNN) with two convolutional layers and one fully connected hidden layer to classify handwritten digits from the MNIST dataset?

**Ans:-** CNN Architecture Overview

Layer Type Details

Input Layer 28×28 grayscale image

Conv2D Layer 1 32 filters, 3×3 kernel, ReLU

MaxPooling2D 2×2 pool size

Conv2D Layer 2 64 filters, 3×3 kernel, ReLU

MaxPooling2D 2×2 pool size

Flatten Convert 2D → 1D

Dense Hidden 128 neurons, ReLU

Output Layer 10 neurons, Softmax

Code:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.utils import to\_categorical

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# Normalize pixel values to [0, 1]

X\_train = X\_train.astype("float32") / 255.0

X\_test = X\_test.astype("float32") / 255.0

# Reshape to add channel dimension (28, 28, 1)

X\_train = X\_train[..., tf.newaxis]

X\_test = X\_test[..., tf.newaxis]

# One-hot encode the labels

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Build the CNN model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.1)

# Evaluate on test data

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy:.4f}')

Output: Test Accuracy: 0.9900