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:

$$L_{binary} = -[y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y})]$$

Where:
- y is the true label (0 or 1)
- $\hat{y}$ 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:

$$L_{categorical} = -\sum_{i=1}^{C} y_i * \log(\hat{y}_i)$$

Where:
- C = number of classes
- $y_i = 1$ if true class is i, else 0
- $\hat{y}_i$ = 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

2. 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

3. 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

4. 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]]

5. 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