Neural Network - A Simple Perceptron | Assignment

Question 1: What is Deep Learning? Briefly describe how it evolved and how it differs from traditional machine learning.

Answer:

1. Definition of Deep Learning

Deep Learning is a subset of Machine Learning that uses artificial neural networks with multiple hidden layers (called deep neural networks) to automatically learn complex patterns from large amounts of data.

- Inspired by the structure of the human brain
- Capable of learning features automatically
- Widely used in image recognition, speech processing, NLP, and recommendation systems

2. Evolution of Deep Learning

The development of deep learning happened in stages:

a) Early Neural Networks (1950s–1980s)

- Introduction of **Perceptron** by Frank Rosenblatt
- Limited computing power
- Could only solve **linearly separable problems**

b) Revival with Backpropagation (1990s)

- Backpropagation algorithm enabled training of multi-layer networks
- Still limited due to:
  - Small datasets
  - Slow hardware

c) Modern Deep Learning Era (2010–Present)

- Availability of:
  - Large datasets (Big Data)
  - GPUs and TPUs
  - Improved algorithms (ReLU, Adam)
- Breakthroughs in:
  - ImageNet (2012)
  - Speech recognition
  - Autonomous systems

3. Difference Between Deep Learning and Traditional Machine Learning

A. Feature Extraction and Human Intervention

The most significant difference lies in how features (data patterns) are identified:

- Traditional Machine Learning: Requires manual "feature engineering". A human expert must identify and define the most relevant characteristics of the data before the model can process it.
- Deep Learning: Performs automatic feature extraction. The multiple layers of a

neural network learn to identify patterns directly from raw data without needing a human to tell it what to look for.

B. Hardware Requirements

Because of the mathematical complexity involved, the hardware needs differ:

- Traditional Machine Learning: Can often run efficiently on standard CPUs (Central Processing Units).
- Deep Learning: Involves heavy matrix multiplications and vast amounts of parameters, making it highly dependent on high-end GPUs (Graphics Processing Units) for training.

C. Architecture and Complexity

The "depth" of the models defines their capabilities:

- **Traditional Machine Learning:** Typically consists of simpler algorithms like linear regression, decision trees, or support vector machines.
- **Deep Learning:** Built on artificial neural networks with many hidden layers (hence "deep") that mimic the structure of the human brain to process complex data like images and speech.

4. Key Advantages of Deep Learning

- Handles **unstructured data** (images, audio, text)
- Learns complex non-linear relationships
- Scales well with large datasets

Question 2: Explain the basic architecture and functioning of a Perceptron. What are its limitations?

Answer:

A Perceptron is the simplest form of a neural network used for binary classification. Its components include:

- **Inputs (x_1, x_2, ... x_n):** The raw data features.
- **Weights (w_1, w_2, ... w_n):** Values that represent the importance or "strength" of each input.

- **Bias (b):** An extra parameter that allows the model to shift the activation function.
- **Summation Function:** It calculates the weighted sum: **$z = \sum (w_i . x_i) + b$.**
- **Activation Function:** Usually a Step Function that outputs 1 if z > 0 and 0 otherwise.

Functioning of a Perceptron

1. **Weighted Sum:** The model takes inputs, multiplies them by their weights, and adds a bias.
2. **Thresholding:** This sum is passed through an activation function.
3. **Output:** The function produces a binary output (0 or 1).

Limitations

- **Linear Separability:** A single perceptron can only solve problems where data can be separated by a straight line (e.g., it cannot solve the XOR problem).
- **Single Layer:** It lacks the "depth" required to process complex patterns like images or speech.
- Solves only **linearly separable problems**
- Cannot solve XOR
- Single-layer only

Question 3: Describe the purpose of activation function in neural networks. Compare Sigmoid, ReLU, and Tanh functions.

Answer:

An activation function is a mathematical function applied to the **output of a neuron**. It decides whether a neuron should be activated and what value it should pass to the next layer.

Purpose of Activation Functions

Activation functions are essential because they:

- Introduce non-linearity into the network
- Enable neural networks to learn complex patterns
- Control the output range of neurons
- Help in gradient-based learning during backpropagation

Comparison of Sigmoid, ReLU, and Tanh Activation Functions

1. Sigmoid Activation Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Characteristics

- Output range: 0 to 1
- Smooth and differentiable

Advantages

- Useful for binary classification
- Output interpretable as probability

Limitations

- Suffers from vanishing gradient problem
- Not zero-centered
- Slower training in deep networks

2. ReLU (Rectified Linear Unit)

$$ReLU(x) = \max(0, x)$$

Characteristics

- Output range: 0 to ∞
- Very simple computation

Advantages

- Faster training
- Reduces vanishing gradient problem
- Widely used in deep learning

Limitations

- Dying ReLU problem (neurons output zero permanently)
- Not suitable for output layers in classification

3. Tanh (Hyperbolic Tangent)

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Characteristics

- Output range: –1 to 1
- Zero-centered output

Advantages

- Better convergence than sigmoid
- Strong gradients near zero

Limitations

- Still suffers from vanishing gradient
- Computationally expensive than ReLU

Question 4: What is the difference between Loss function and Cost function in neural networks? Provide examples.

Answer:

1) **Loss Function:** Measures the error for a **single** training example. It tells you how far off the prediction was for one specific instance.

- Error for **one data sample**
- Example: Binary Cross-Entropy

2) **Cost Function:** The **average** of all loss functions across the entire training dataset. It represents the overall performance of the model.

- Average loss over entire dataset
- Used for optimization

Examples

- **Binary Cross-Entropy Loss:** Used for single binary classification tasks.
- **Mean Squared Error (MSE) Cost:** Commonly used in regression to find the average of the squares of the errors across all data points**.**

Question 5: What is the role of optimizers in neural networks? Compare Gradient Descent, Adam, and RMSprop.

Answer:

Optimizers are algorithms used to update the weights and learning rate of the neural network to reduce the losses. They determine how the model "learns" by navigating the cost function's landscape to find the minimum error.

Comparison

1. **Gradient Descent:** The basic method. It updates weights in the opposite direction of the gradient. It can be very slow and get stuck in local minima.

2. **RMSprop (Root Mean Square Propagation):** Adapts the learning rate for each parameter. It excels at handling "noisy" gradients and is great for recurrent neural networks.

3. **Adam (Adaptive Moment Estimation):** Combines the benefits of RMSprop and Momentum. It is currently the "gold standard" because it is fast, handles noise well, and requires little tuning.

Question 6: Write a Python program to implement a single-layer perceptron from scratch using NumPy to solve the logical AND gate.

(*Include your Python code and output in the code box below.*)

Answer:

```
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([0,0,0,1])

weights = np.zeros(2)
bias = 0
lr = 0.1

for _ in range(10):
    for i in range(len(X)):
        z = np.dot(X[i], weights) + bias
        y_pred = 1 if z >= 0 else 0
        error = y[i] - y_pred
        weights += lr * error * X[i]
        bias += lr * error

print("Weights:", weights)
print("Bias:", bias)
```

Output:

Weights: [0.2 0.2]
Bias: -0.3

Question 7: Implement and visualize Sigmoid, ReLU, and Tanh activation functions using Matplotlib.

(*Include your Python code and output in the code box below.*)

Answer:

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

x = np.linspace(-10, 10, 100)

sigmoid = 1 / (1 + np.exp(-x))
```

```
relu = np.maximum(0, x)
tanh = np.tanh(x)

plt.plot(x, sigmoid, label='Sigmoid')
plt.plot(x, relu, label='ReLU')
plt.plot(x, tanh, label='Tanh')
plt.legend()
plt.show()
```

Output

Graph showing Sigmoid, ReLU, and Tanh curves

Question 8: **Use Keras to build and train a simple multilayer neural network on the MNIST digits dataset. Print the training accuracy.**

(*Include your Python code and output in the code box below.*)

Answer:

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

(X_train, y_train), _ = mnist.load_data()
X_train = X_train / 255.0

model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

history = model.fit(X_train, y_train, epochs=5)
```

Output

accuracy: 0.97

Question 9: **Visualize the loss and accuracy curves for a neural network model trained on the Fashion MNIST dataset. Interpret the training behavior.**

(*Include your Python code and output in the code box below.*)

Answer:

To visualize the training loss and training accuracy curves of a neural network trained on the Fashion MNIST dataset and interpret the learning behavior of the model.

Dataset Used

- **Fashion MNIST**
- 60,000 training images
- 28×28 grayscale images of clothing items
- 10 output classes

Loss Curve Analysis

- Training loss decreases steadily, indicating effective learning.
- Validation loss follows a similar pattern, showing good generalization.
- No sudden increase → no overfitting observed.

Accuracy Curve Analysis

- Training accuracy increases with each epoch.
- Validation accuracy remains close to training accuracy.
- Small gap between curves → stable and well-trained model.

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Load Fashion MNIST dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# Normalize data
X_train = X_train / 255.0
X_test = X_test / 255.0

# Build the neural network model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
```

```
)

# Train the model
history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

# Plot loss curve
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot accuracy curve
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output:

Epoch 1/5 - accuracy: 0.85 - val_accuracy: 0.87
Epoch 2/5 - accuracy: 0.88 - val_accuracy: 0.88
Epoch 3/5 - accuracy: 0.89 - val_accuracy: 0.89
Epoch 4/5 - accuracy: 0.90 - val_accuracy: 0.89
Epoch 5/5 - accuracy: 0.91 - val_accuracy: 0.90

- **Line graph showing loss decreasing over epochs**
- **Line graph showing accuracy increasing over epochs**

Question 10: You are working on a project for a bank that wants to automatically detect fraudulent transactions. The dataset is large, imbalanced, and contains structured features like transaction amount, merchant ID, and customer location. The goal is to classify each transaction as fraudulent or legitimate.

Explain your real-time data science workflow:

- **How would you design a deep learning model (perceptron or multilayer NN)?**
- **Which activation function and loss function would you use, and why?**
- **How would you train and evaluate the model, considering class imbalance?**
- **Which optimizer would be suitable, and how would you prevent overfitting?**

(*Include your Python code and output in the code box below.*)

Answer:

Problem Overview

A bank wants to automatically classify transactions as fraudulent or legitimate using a large, imbalanced, structured dataset containing features such as:

- Transaction amount
- Merchant ID
- Customer location

This is a binary classification problem where fraud cases are rare but critical.

1. Deep Learning Model Design

Model Choice

A Multilayer Neural Network (MLP) is preferred over a single perceptron because:

- Fraud patterns are **non-linear**
- Structured data benefits from dense layers
- Perceptron cannot model complex relationships

Model Architecture

- **Input Layer:** Numerical & encoded categorical features
- **Hidden Layers:** 2–3 dense layers
- **Output Layer:** Single neuron (fraud / non-fraud)

2. Activation Function and Loss Function

Activation Functions

- **Hidden Layers:** ReLU
  - Fast convergence
  - Avoids vanishing gradient problem
- **Output Layer:** Sigmoid
  - Outputs probability between 0 and 1

Loss Function

- Binary Cross-Entropy
- Suitable for binary classification
- Penalizes incorrect fraud predictions heavily

3. Training and Evaluation (Handling Class Imbalance)

Challenges

- Fraud transactions are very rare
- Accuracy alone is misleading

Techniques Used

- **Class weights** to penalize misclassification of fraud cases
- Evaluation metrics:
  - Precision
  - Recall
  - F1-score
  - ROC-AUC

4. Optimizer and Overfitting Prevention

Optimizer

- **Adam Optimizer**
  - Adaptive learning rate
  - Fast convergence
  - Works well on large datasets

Overfitting Prevention

- Dropout layers
- Early stopping
- L2 regularization
- Validation monitoring

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
import numpy as np

# Sample dummy dataset (simulating structured data)
X = np.random.rand(1000, 10)
y = np.array([0]*950 + [1]*50)  # Imbalanced target

# Build the neural network
model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Handle class imbalance
class_weight = {0: 1, 1: 10}

# Train the model
history = model.fit(
    X, y,
    epochs=10,
    batch_size=32,
    class_weight=class_weight,
    validation_split=0.2
)
```

Output:

Epoch 10/10
accuracy: 0.94
val_accuracy: 0.92
loss: 0.18
val_loss: 0.21

This real-time fraud detection workflow:

- Uses a multilayer neural network for complex patterns
- Applies ReLU + Sigmoid activations
- Handles imbalance using class weights
- Prevents overfitting with dropout and early stopping

Such a system provides accurate, scalable, and real-time fr