Training Day-73 Report:

Networks in Python and TensorFlow Basics under the context of Deep Learning:

Networks in Python

In deep learning, a network refers to an artificial neural network (ANN), which consists of layers of interconnected nodes (neurons) for feature extraction and prediction.

Building a Neural Network in Python (from scratch)

Below is an example of a simple feedforward neural network:

```
import numpy as np


# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivative of sigmoid
def sigmoid_derivative(x):
    return x * (1 - x)


# Input data (XOR problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])


# Initialize weights and biases
np.random.seed(42)
weights_input_hidden = np.random.rand(2, 2)  # 2 input nodes, 2 hidden nodes
weights_hidden_output = np.random.rand(2, 1) # 2 hidden nodes, 1 output node
bias_hidden = np.random.rand(1, 2)
bias_output = np.random.rand(1, 1)
```

```
# Training parameters
epochs = 10000
learning_rate = 0.1

for epoch in range(epochs):
    # Forward pass
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    final_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)

    # Backward pass (calculate gradients)
    error = y - final_output
    d_output = error * sigmoid_derivative(final_output)
    d_hidden_layer = d_output.dot(weights_hidden_output.T) *
sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_hidden_output += hidden_layer_output.T.dot(d_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_output += np.sum(d_output, axis=0) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0) * learning_rate

print("Output after training:")
print(final_output)
```

This example demonstrates the structure and operations of a neural network.

TensorFlow Basics

TensorFlow is an open-source library widely used for building and training machine learning and deep learning models.

Core Components

1. Tensors: Multidimensional arrays (like NumPy arrays) that flow through the network.
2. Graphs: Computational graphs represent operations and the flow of tensors.
3. Operations (Ops): Nodes in the computational graph.

Basic Workflow in TensorFlow

1. Import TensorFlow:
2. import tensorflow as tf
3. Define a Neural Network: Example: A simple feedforward neural network for classification.
4. import tensorflow as tf
5. from tensorflow.keras.models import Sequential
6. from tensorflow.keras.layers import Dense
7.
8. # Create a sequential model
9. model = Sequential([
10.   Dense(32, activation='relu', input_shape=(2,)), # Input layer with 32 neurons
11.   Dense(16, activation='relu'), # Hidden layer with 16 neurons
12.   Dense(1, activation='sigmoid') # Output layer with sigmoid for binary classification
13.])
14.
15. # Compile the model
16. model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
17.
18. # Dummy dataset (XOR problem)
19. X = [[0, 0], [0, 1], [1, 0], [1, 1]]
20. y = [0, 1, 1, 0]
21.
22. # Train the model
23. model.fit(X, y, epochs=100, verbose=0)
24.
25. # Evaluate the model

26. print("Model evaluation:", model.evaluate(X, y))
27. Key TensorFlow APIs:
    - tf.keras: High-level API for building models.
    - tf.data: Tools for creating efficient input pipelines.
    - tf.function: For converting Python functions into TensorFlow graphs.
28. Building Custom Models: Use the TensorFlow low-level API to define your model architecture and gradients manually.
29. class CustomModel(tf.keras.Model):
30.   def __init__(self):
31.   super(CustomModel, self).__init__()
32.   self.hidden_layer = tf.keras.layers.Dense(32, activation='relu')
33.   self.output_layer = tf.keras.layers.Dense(1, activation='sigmoid')
34.
35.   def call(self, inputs):
36.   x = self.hidden_layer(inputs)
37.   return self.output_layer(x)
38.
39. # Instantiate and compile the custom model
40. model = CustomModel()
41. model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])