**CLO 1: Recall and Understand Foundational Concepts**

**Topics: Foundations of Neural Networks and Deep Learning**

* **Question 1**: *(Remember)*  
  Define the Universal Approximation Theorem and explain its significance in neural networks. *(Marks: 5)*

The Universal Approximation Theorem states that a neural network with at least one hidden layer of a sufficient number of neurons, and a non-linear activation function can approximate any continous function to an arbitrary level of accuracy.

It is a fundamental theorem in neural network theory, which is why neural networks are called universal approximators.

* **Question 2**: *(Understand)*  
  Describe the role of each component in a neural network (weights, biases, activation functions). How do these components collectively enable learning? *(Marks: 10)*

The weight of the connection affects how much input is passed between neurons.

Bias, a constant value that changes is added to the previous computation involving weight.

Activation function determine which neurons are activated as information passes through network.

During the forward pass, inputs are process through these components to produce an output. The error between predicted and true values in then used to adjust weights and biases via backpropogation, improving network’s accuracy overtime

**CLO 2: Apply Linear Algebra Techniques to Neural Networks**

**Topics: Linear Algebra Essentials for Deep Learning**

* **Question 3**: *(Apply)*  
  Given a 2D transformation matrix , apply this transformation to the vector, and interpret the result in the context of neural network transformations. *(Marks:10)*

[4​

7]

* **Question 4**: *(Apply)*  
  Demonstrate how matrix multiplication is used to compute the forward pass in a single-layer neural network with inputs and weights W. *(Marks: 10)*

WT is the transpose of the weight vector, and is the multiplied by input vector x with added bias which gives the weighted sum z

Z = WT \* x + b

**CLO 3: Analyze Optimization Techniques in Training**

**Topics: Gradient Descent and Optimization**

* **Question 5**: *(Analyze)*  
  Consider a simple quadratic loss function , where . Analyze how gradient descent updates the parameters w and b to minimize L. Include partial derivative computations. *(Marks: 15)*

**With respect to w:** ∂L/∂w=−2x(y−(wx+b)) **With respect to b:** ∂L/∂b=−2(y−(wx+b))

**Update for w:** w←w+2ηx(y−y^) **Update for b:** b←b+2η(y−y^) where η is the learning rate

* **Question 6**: *(Understand)*  
  Differentiate between batch gradient descent, stochastic gradient descent (SGD), and mini-batch gradient descent, citing their pros and cons in terms of computational efficiency and convergence. *(Marks: 10)*

**BGD:** Entire dataset / Accurate gradient estimates, smooth convergence / require significant memory

**SGD:** Single data point / Fast updates, large datasets, better generalization / high variance, slower convergence

**Mini-Batch:** Subset of dataset / Balances speed & accuracy, utilizes parallelism / requires tuning of learning rate

**CLO 4: Evaluate the Role of CNNs in Image Recognition**

**Topics: Convolutional Neural Networks (CNNs)**

* **Question 7**: *(Evaluate)*  
  Compare the performance of convolutional layers and fully connected layers in image recognition tasks. Which is better for spatial data, and why? *(Marks: 10)*

**Convolutional Layers** are better for image recognition tasks as they preserve spatial relationships and effectively learn hierarchical features from local patterns.

**Fully Connected Layers** do not maintain spatial structure and treat each input independently, making them less suitable for processing spatial data like images.

**Conclusion:** Convolutional layers are more appropriate for spatial data because they can capture spatial hierarchies, while fully connected layers are more suited for combining abstracted features in the final stages.

* **Question 8**: *(Understand)*  
  Discuss the importance of pooling layers in CNNs. How do max pooling and average pooling differ in their effects on feature extraction? *(Marks: 10)*

**Importance:** Pooling layers reduce feature map size, decrease computation, prevent overfitting, and provide translation invariance.

**Max Pooling:** Retains the strongest feature in each region, emphasizing prominent features and improving feature robustness.

**Average Pooling:** Averages values in each region, resulting in smoother feature maps and capturing broader patterns but losing fine details.

**Conclusion:** Max pooling is better for emphasizing strong features, while average pooling provides a more generalized representation.

**CLO 5: Create and Experiment with Neural Networks**

**Cumulative Question Covering All Topics**

* **Question 9**: *(Create)*  
  Design a CNN architecture using PyTorch for a digit classification task (e.g., MNIST dataset). Specify the number of layers, activation functions, and loss function used, and include the training process. *(Marks: 20)*

import torch

import torch.nn as nn

import torch.optim as optim

# Define CNN architecture

class CNN(nn.Module):

def \_\_init\_\_(self):

super(CNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, activation=nn.ReLU())

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(32 \* 13 \* 13, 10) # Assuming input size 28x28

def forward(self, x):

x = self.pool(self.conv1(x))

x = x.view(-1, 32 \* 13 \* 13)

x = self.fc1(x)

return x

# Instantiate model

**model = CNN()**

**criterion = nn.CrossEntropy()**

**Optimizer = optim.Adam(model.parameters(), lr=0.001)**

# Training loop

for epoch in range(10):

for inputs, labels in train\_loader:

optimizer.zero\_grad()

**outputs = model(inputs)**

**loss = criterion(outputs, labels)**

**loss.backward()**

**optimizer.step()**

**running\_loss += loss.item()**

**print(f"Epoch {epoch+1}, Loss: {running\_loss/len(train\_loader):.4f}")**