Question-1: Which loss function, out of Cross-Entropy and Mean Squared Error, works best with logistic regression because it guarantees a single best answer (no room for confusion)? Explain why this is important and maybe even show how it affects the model's training process. [3 Marks] [Theory]

Ans.- The cross-entropy loss function is typically preferred over Mean Squared Error (MSE) for logistic regression. The reason lies in the nature of logistic regression and the desired output it generates.

In logistic regression, the output is a probability value between 0 and 1, representing the likelihood of a particular instance belonging to a certain class. This output is obtained by passing the weighted sum of features through a sigmoid function. The output can be interpreted as the probability of the instance belonging to the positive class.

Cross Entropy loss, also known as log loss, is well-suited for logistic regression because it measures the difference between the predicted probabilities and the actual labels. It calculates the loss as the negative log-likelihood of the observed data given the model's predictions. Mathematically, it's defined as:

$$H(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

Where:

- y is the true label (0 or 1).
- $\hat{y}$ is the predicted probability.
- N is the number of samples.

Cross Entropy loss ensures that the model aims to predict the probabilities as close to the true labels as possible. It penalizes the model more heavily for confidently incorrect predictions, thus pushing the model to learn better representations.

On the other hand, Mean Squared Error (MSE) loss measures the squared difference between the predicted values and the true labels. While it's a suitable loss function for regression tasks where the output is continuous, it's not ideal for logistic regression because it doesn't directly optimize for probabilities. Using MSE with logistic regression can lead to suboptimal results as it may not effectively capture the true goal of the classification task.

In summary, Cross Entropy loss ensures that logistic regression models produce probabilities that are as close as possible to the true labels, making it the preferred choice for such tasks. This directly aligns with the objective of logistic regression, which is to classify instances into one of two classes based on their probabilities.

Question-2: For a binary classification task with a deep neural network (containing at least one hidden layer) equipped with linear activation functions, which of the following loss functions guarantees a convex optimization problem? Justify your answer with a formal proof or a clear argument. (a) CE (b) MSE (c) Both (A) and (B) (d) None [3 Marks] [Theory]

Ans.- When considering a deep neural network (DNN) configured for binary classification with linear activation functions, the Mean Squared Error (MSE) loss function emerges as the optimal choice, guaranteeing convex optimization. The linearity introduced by the use of linear activation functions in hidden layers ensures a convex optimization landscape across the entire network, facilitating efficient parameter optimization. Additionally, the inherent convexity of the MSE loss function, driven by its squared term, further reinforces the convex nature of the optimization problem, as deviations from true labels are penalized in a convex manner. This convexity guarantees that any local minimum encountered during optimization using techniques like gradient descent is also a global minimum, ensuring convergence to the optimal solution.

Conversely, the Cross-Entropy (CE) loss function, commonly utilized in classification tasks, lacks the guarantee of convexity, particularly in scenarios where linear activation functions are predominant. Although CE loss functions excel with nonlinear activation functions like sigmoid or softmax, their convexity hinges on such choices. Thus, in contexts where linear activation functions are employed, the CE loss function may fail to ensure convex optimization. Consequently, in the specified setting, the

MSE loss function stands out as the preferred option, offering convex optimization and providing assurance of convergence to the global minimum, essential for the effective training of deep neural networks.
So, the answer will be **(B)** option.

Question-3: Dense Neural Network: Implement a feedforward neural network with dense layers only. Specify the number of hidden layers, neurons per layer, and activation functions. How will you preprocess the input images? Consider hyperparameter tuning strategies. [2 for implementation and 3 for explanation] [Code and Report]

Ans.- The code file is given in the repository.

Report: Hyperparameter Tuning for Convolutional Neural Network on MNIST Dataset

Objective

The objective of this assignment was to develop and tune a convolutional neural network (CNN) for classifying handwritten digits from the MNIST dataset. The primary focus was on exploring hyperparameter tuning strategies and preprocessing techniques to enhance model performance.

Dataset

The MNIST dataset consists of grayscale images of handwritten digits (0-9). It is commonly used for digit classification tasks and contains 60,000 training images and 10,000 test images.

Preprocessing

1. **Normalization:** Pixel values of the input images were normalized to the range [0, 1].
2. **Reshaping:** Input images were reshaped to include a channel dimension, suitable for processing by a CNN.

Convolutional Neural Network Architecture

The CNN architecture comprised multiple convolutional and pooling layers followed by fully connected layers for classification. The architecture was defined as follows:

- Convolutional layer: 32 filters, kernel size (3, 3), ReLU activation
- MaxPooling layer: Pool size (2, 2)
- Convolutional layer: 64 filters, kernel size (3, 3), ReLU activation
- MaxPooling layer: Pool size (2, 2)
- Flatten layer
- Dense layer: 128 neurons, ReLU activation
- Output layer: 10 neurons (softmax activation)
- Here we used Flatten layer, Dense layer, and Output layer.

Hyperparameters Tuned

1. Number of neurons in the dense layer: 32, 64, 128.

2. Activation function for the dense layer: ReLU, tanh, sigmoid.

Hyperparameter Tuning Strategy

Grid search was employed using scikit-learn's `GridSearchCV` with 3-fold cross-validation. The grid search explored various combinations of neurons in the dense layer and activation functions. The model's performance was evaluated based on the mean validation accuracy across folds.

Results

- The grid search identified the best model configuration with an overall validation accuracy of 94.64%. This model utilized ReLU activation with 128 neurons in the dense layer.

- ReLU activation consistently outperformed tanh and sigmoid activation functions across different numbers of neurons in the dense layer.

- Increasing the number of neurons generally improved performance, with the highest accuracy achieved using 128 neurons.

- The test accuracy of the best model on the unseen test set was 95.74%.

Conclusion

Preprocessing techniques such as normalization and reshaping, coupled with hyperparameter tuning, significantly improved the CNN's performance on the MNIST dataset. The selected model demonstrated competitive accuracy, highlighting the effectiveness of grid search in optimizing CNN architectures.

Recommendations

Based on the results, it is recommended to use ReLU activation with a higher number of neurons (128) in the dense layer for better classification performance on the MNIST dataset. Further exploration of hyperparameters and architectural modifications may yield even better results.

Question-4: Build a classifier for Street View House Numbers (SVHN) (Dataset) using pre-trained model weights from PyTorch. Try multiple models like LeNet-5, AlexNet, VGG, or ResNet(18, 50, 101). Compare performance comments on why a particular model is well suited for the SVHN dataset. (You can use a subset of the dataset (25%) in case you do not have enough compute.) [4 Marks] [Code and Report]

Ans.- The code file is given in the repository.

Report: Comparison of Pretrained Models for SVHN Dataset Classification

Introduction

The Street View House Numbers (SVHN) dataset is a real-world dataset consisting of images of house numbers collected from Google Street View. The task is to classify these images into their respective digit labels. In this report, we compare the performance of various pre-trained models including LeNet-5, AlexNet, VGG, ResNet-18, ResNet-50, and ResNet-101 on the SVHN dataset.

Experimental Setup

We experimented with the SVHN dataset using PyTorch and pre-trained models available in torchvision. The dataset was preprocessed, and a subset consisting of 25% of the data was used for training and testing due to computational constraints.

Performance Metrics

We evaluated the performance of each model using the following metrics:

- Test Accuracy: The percentage of correctly classified images in the test set.
- Precision: The ability of the classifier not to label a negative sample as positive.
- Recall: The ability of the classifier to find all positive samples.
- F1-score: The harmonic mean of precision and recall, providing a balance between the two metrics.

Results

Model	Test Accuracy	Precision	Recall	F1-score
LeNet-5	77.74%	0.769	0.751	0.757
VGG-16	83.19%	0.830	0.814	0.814
ResNet-18	88.58%	0.877	0.876	0.875
ResNet-50	87.12%	0.863	0.862	0.861
ResNet-101	79.61%	0.800	0.778	0.777

Analysis

- **LeNet-5:** LeNet-5 achieved a test accuracy of 77.74%. While it performed reasonably well, its performance is relatively lower compared to deeper architectures like VGG and ResNet.
- **VGG-16:** VGG-16 outperformed LeNet-5 with a test accuracy of 83.19%. Its deeper architecture and increased capacity allowed it to learn more complex features, resulting in improved performance.
- **ResNet-18 and ResNet-50:** Both ResNet-18 and ResNet-50 achieved higher test accuracies compared to LeNet-5 and VGG-16. ResNet's skip connections and deeper architecture enabled better feature propagation and alleviated the vanishing gradient problem, leading to superior performance.

- **ResNet-101:** ResNet-101 achieved a test accuracy of 79.61%, which is slightly lower than ResNet-18 and ResNet-50. While ResNet-101 has a deeper architecture, it might suffer from overfitting due to its increased complexity.

Conclusion

Among the models tested, ResNet-18 and ResNet-50 demonstrated the best performance on the SVHN dataset. Their ability to effectively propagate features through skip connections and handle deeper architectures made them well-suited for this task. While VGG-16 also performed well, its performance was slightly inferior to ResNet models. LeNet-5, although a classic architecture, lagged behind the deeper models due to its limited capacity to learn complex features. Overall, the choice of a pre-trained model for the SVHN dataset should consider both performance and computational efficiency, with ResNet-18 being a strong candidate for its balance of accuracy and model complexity.