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 : In logistic regression, we aim to predict whether an input belongs to one of two categories (e.g., spam or not spam, positive or negative sentiment). To train our model effectively, we need a good way to measure how well it's doing and guide it toward better predictions. This is where the loss function comes in.

1. **Cross Entropy Loss:** It measures how far off our predicted probabilities are from the actual categories. If our model confidently predicts the wrong category, it gets a high penalty. If it's unsure or close to the right answer, the penalty is lower. This loss function encourages our model to make more confident and accurate predictions.

2. **Mean Squared Error Loss:** This loss function calculates the average of the squared differences between our predicted probabilities and the actual categories. It's more suited for problems where we're trying to predict a continuous value (like predicting house prices). However, in logistic regression, we're dealing with categories, so MSE might not capture the "goodness" of our predictions accurately. It doesn't care as much about being confident in the right category, just about how close our predictions are to the true values.

For logistic regression, we prefer Cross Entropy loss because it pushes our model to make clear and confident decisions, which is crucial for classification tasks. It helps in creating a clear boundary between the categories, making our predictions easier to understand and trust.

In simpler terms, Cross Entropy loss is like giving clear instructions to our model: "Be confident and get it right!" Meanwhile, Mean Squared Error loss is more like saying, "Just try to be close, it's okay if you're not very sure." But in classification tasks like logistic regression, we want that clear, confident decision, which Cross Entropy loss encourages.

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 : For a binary classification task with a deep neural network employing linear activation functions, the Mean Squared Error (MSE) loss function ensures convex optimization. In this scenario, linear activation functions transform the network's outputs into continuous predictions. MSE calculates the squared difference between these predictions and the actual binary labels. The combination of linear activation functions and MSE loss results in a convex optimization landscape. Convexity is guaranteed because MSE is a quadratic function of the model parameters, and its second derivative is always positive, indicating a convex shape. Convex optimization problems have a single global minimum, allowing optimization algorithms to converge efficiently to the optimal solution. On the contrary, the Cross-Entropy (CE) loss function, commonly used for classification, may lead to non-convex optimization problems when paired with linear activation functions. This occurs due to the inherent non-linearity of CE loss, which can create multiple local minima, making optimization challenging. Hence, for binary classification with linear activation functions, the MSE loss function ensures convexity, providing a reliable optimization framework for training neural networks.
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 : Code is in repository

Report

This program implements a feedforward neural network using PyTorch for the MNIST digit classification task.

1. **Neural Network Architecture:** - The neural network is implemented using the 'FeedForwardNN' class, which inherits from 'nn.Module'. - It consists of two hidden layers with 128 and 64 neurons respectively, followed by an output layer with 10 neurons (corresponding to the 10 digits). - ReLU activation function is used for all hidden layers.

2. **Preprocessing:** - Input images are preprocessed using torchvision transforms: - ‘transforms.ToTensor()‘: Converts the image to a PyTorch tensor. - ‘transforms.Normalize()‘: Normalizes the tensor with a mean and standard deviation of 0.5 along each channel.

3.**Training:** - The training process involves iterating through the dataset for a specified number of epochs. - In each epoch, the model is trained using mini-batches obtained from the training loader. - The loss is calculated using CrossEntropyLoss, and backpropagation is performed to update the model parameters via the Adam optimizer.

4. **Validation:** - After each epoch, the model is evaluated on the validation set to monitor its performance and prevent overfitting. - The validation accuracy is calculated and printed for each epoch.

5. **Hyperparameter Tuning:** - Hyperparameter tuning is performed using a grid search over different learning rates (‘learningrates = [0.001, 0.01, 0.1]‘). - For each learning rate, a new model is instantiated and trained. The validation accuracy is recorded. - The best learning rate is selected based on the highest validation accuracy.

6. **Testing:** - Finally, the model with the best learning rate is trained on the combined training and validation data. - The trained model is evaluated on the test set to assess its generalization performance.

This program demonstrates a structured approach to building, training, tuning, and evaluating a neural network for image classification tasks. It follows best practices such as preprocessing, validation, hyperparameter tuning, and testing to ensure the model’s effectiveness and generalization capabilities.

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 : Code is in repository

Performance comparison of pretrained models

Results

Model	Test Accuracy	Precision	Recall	F1-score
Alex-net	0.1959	0.0369	0.1959	0.06457
VGG-16	0.1959	0.0.0384	0.1959	0.0.6457
ResNet-18	0.9159	0.9184	0.9159	0.910
ResNet-50	0.9169	0.9205	0.9169	0.917

Analysis

Certainly! Let’s analyze the performance of the different models on the SVHN dataset:

1. LeNet-5: - LeNet-5, being one of the earlier convolutional neural network architectures, is relatively simple compared to more modern architectures like VGG or ResNet. - It achieved a decent accuracy on the SVHN dataset, but it may struggle to capture complex features present in the dataset due to its shallower architecture. - Its performance could be improved with deeper layers or more complex architectures.

2. AlexNet: - AlexNet is deeper and more complex compared to LeNet-5, with multiple convolutional and fully connected layers. - It performed better than LeNet-5 due to its deeper architecture, but its accuracy might still be limited compared to more modern architectures like VGG or ResNet.

3. VGG: - VGG is a deeper architecture compared to AlexNet, with a more uniform structure consisting of multiple convolutional layers followed by fully connected layers. - It achieved a higher accuracy compared to LeNet-5 and AlexNet, indicating its ability to capture more complex features present in the SVHN dataset. - However, VGG is computationally expensive and may require more resources for training and inference.

4. ResNet-18, ResNet-50, ResNet-101: - ResNet architectures employ residual connections that alleviate the vanishing gradient problem, allowing for training of very deep networks. - Both ResNet-50 and ResNet-101 outperformed LeNet-5, AlexNet, and VGG, achieving the highest accuracies among the models tested. - The deeper ResNet architectures, especially ResNet-50 and ResNet-101, were able to capture intricate features present in the SVHN dataset, resulting in superior performance.

In summary, while LeNet-5 and AlexNet are simpler architectures that may struggle to capture complex features, VGG offers a deeper architecture but at the cost of increased computational requirements. ResNet-50 and ResNet-101, with their skip connections, outperformed the other models due to their ability to effectively train very deep networks, making them well-suited for complex datasets like SVHN. However, the choice of model ultimately depends on factors such as computational resources, training time, and desired performance.