Deep Learning - Assignment 1

Name:	**Sumit Kumar**
Registration No./Roll No.:	2311006
Institute/University Name:	IISER Bhopal
Program/Stream:	Ph.D./DSE

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.

In logistic regression, the choice of loss function plays a crucial role in determining the model's performance and optimization process. Two common loss functions used in logistic regression are Cross Entropy Loss and Mean Squared Error (MSE) Loss. Let's discuss why Cross Entropy Loss is generally preferred over MSE Loss for logistic regression tasks.

1.1 Cross Entropy Loss

Cross Entropy Loss is a popular choice for classification tasks, including logistic regression. Here's why it works best:

- **Probability Interpretation**: Logistic regression outputs probabilities between 0 and 1, representing the likelihood of an instance belonging to a certain class. Cross Entropy Loss directly measures the difference between these predicted probabilities and the true labels, making it suitable for evaluating classification performance.
- **Sensitivity to Probability Estimates**: Cross Entropy Loss penalizes the model based on the difference between predicted probabilities and the true labels. It heavily penalizes confidently wrong predictions, ensuring that the model learns to assign accurate probabilities.
- **Optimization Objectives**: The goal of logistic regression is to maximize the likelihood of correctly predicting class labels. Minimizing Cross Entropy Loss aligns with this objective, as it encourages the model to adjust its parameters to better match the true distribution of the data.
- **Mathematical Properties**: Cross Entropy Loss is convex and continuously differentiable, making it well-suited for optimization algorithms. It facilitates efficient optimization using gradient-based methods like gradient descent.
- **No Room for Confusion**: Cross Entropy Loss encourages the model to output probabilities close to 1 for instances belonging to the positive class and close to 0 for instances belonging to the negative class. This clear distinction between classes ensures that there is no room for confusion, as the model aims to assign a single best answer for each instance.

1.2 Mean Squared Error (MSE) Loss

MSE Loss is primarily used for regression tasks rather than classification tasks like logistic regression. Here's why it is less suitable for logistic regression:

- **Interpretation**: MSE measures the average squared difference between predicted and true values, which may not be well-aligned with the objectives of logistic regression. It does not directly model probabilities, making it less appropriate for binary classification tasks.
- **Optimization**: Using MSE in logistic regression may result in slower convergence and suboptimal performance compared to Cross Entropy Loss. It may not provide meaningful feedback to the model during training, leading to less accurate classifications.

In summary, Cross Entropy Loss is the preferred choice for logistic regression due to its suitability for classification tasks, sensitivity to probability estimates, alignment with optimization objectives, mathematical properties, and ability to provide clear distinctions between classes.

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.

In a binary classification task with a deep neural network equipped with linear activation functions, the Cross Entropy (CE) loss function guarantees a convex optimization problem, while the Mean Squared Error (MSE) loss function does not.

Here's the justification:

Cross Entropy (CE) Loss Function:

The CE loss function is defined as

$$\text{CE} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right)$$

where y_i is the true label (either 0 or 1) and $\hat{y}_i$ is the predicted probability of the positive class.

- The CE loss function is convex because both the logarithmic and exponential functions are convex.
- The convexity of the CE loss function ensures that the optimization problem, when minimizing the CE loss, is convex, leading to convergence to the global minimum.

Mean Squared Error (MSE) Loss Function:

The MSE loss function is defined as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the true label (either 0 or 1) and $\hat{y}_i$ is the predicted output of the neural network.

- The MSE loss function is not guaranteed to be convex when applied to binary classification tasks with linear activation functions.
- The reason for this non-convexity is that the squared term in the MSE loss function can lead to multiple local minima, making it challenging to ensure convergence to the global minimum.

Given the above analysis:

- Option (a) CE: The Cross Entropy (CE) loss function guarantees a convex optimization problem for binary classification tasks with linear activation functions.
- Option (b) MSE: The Mean Squared Error (MSE) loss function does not guarantee a convex optimization problem for such tasks.

- Option (c) Both (A) and (B): This option is incorrect because only the CE loss function satisfies the requirement of guaranteeing a convex optimization problem.
- Option (d) None: This option is also incorrect because the CE loss function meets the criterion.

Therefore, the correct answer is (a) CE.

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.

3.1 Introduction

The provided code implements a dense neural network (DNN) for classifying handwritten digits from the MNIST dataset. MNIST is a benchmark dataset commonly used for evaluating machine learning models, particularly for image classification tasks. The DNN architecture consists of densely connected layers, allowing for complex nonlinear mappings between input features and output predictions.

3.2 Dataset Preparation

The MNIST dataset comprises grayscale images of handwritten digits, each of size 28x28 pixels. Before feeding the images into the neural network, they are preprocessed using the following steps:

- **Conversion to Tensors**: Images are converted from PIL format to PyTorch tensors, which are the preferred data type for working with neural networks in PyTorch.
- **Normalization**: Pixel values are normalized to the range [-1, 1] to ensure numerical stability during training. This is achieved by subtracting the mean (0.5) and dividing by the standard deviation (0.5) of the pixel values.

3.3 Dense Neural Network Architecture

The DNN architecture is defined by the `DenseNet` class, which inherits from `nn.Module` in PyTorch. Here are the key components of the DNN architecture:

- **Flatten Layer**: The input images, originally 28x28 pixels, are flattened into a 1D tensor of size 784 (28*28) before being passed to the first hidden layer.
- **Hidden Layers**: The DNN consists of a configurable number of hidden layers, each containing densely connected (fully connected) neurons. The number of hidden layers, neurons per layer, and activation function are specified as hyperparameters.
- **Activation Functions**: Two types of activation functions are supported: ReLU (Rectified Linear Unit) and Sigmoid. ReLU is commonly used for its simplicity and effectiveness in combating the vanishing gradient problem, while Sigmoid is useful for binary classification tasks.
- **Output Layer**: The output layer consists of neurons corresponding to the number of output classes (10 for MNIST digits). The LogSoftmax activation function is applied to generate class probabilities, and the class with the highest probability is selected as the predicted digit.

3.4 Training

The model is trained using the provided trainmodel function, which iterates over the specified number of epochs. During each epoch, the training data is processed in batches using a DataLoader. The training loop involves the following steps:

- **Forward Pass**: Input images are passed through the neural network to obtain output predictions.
- **Loss Calculation**: The negative log-likelihood loss (NLLLoss) is computed between the predicted class probabilities and the ground truth labels.
- **Backpropagation**: Gradients of the loss function with respect to the model parameters are computed using backpropagation.
- **Parameter Updates**: The optimizer (Adam) adjusts the model parameters based on the computed gradients to minimize the loss function.

3.5 Evaluation

After training, the model is evaluated on the test set using the evaluatemodel function. Test accuracy, defined as the percentage of correctly classified images, is calculated to assess the performance of the trained model.

3.6 Conclusion

In summary, the dense neural network implemented in this code demonstrates the effectiveness of densely connected layers in learning complex patterns from image data. By training on the MNIST dataset, the model achieves competitive accuracy in classifying handwritten digits. Further experimentation with hyperparameters, network architecture, and optimization techniques could potentially improve model performance and generalization capabilities.

4 Build a classifier for Street View House Numbers (SVHN) (Dataset) using pretrained model weights from PyTorch. Try multiple models like LeNet-5, AlexNet, VGG, or ResNet(18, 50, 101). Compare performance comment why a particular model is well suited for SVHN dataset.

4.1 Introduction

This project aimed to assess the performance of various pretrained convolutional neural network (CNN) models on the Street View House Numbers (SVHN) dataset. The SVHN dataset consists of labeled digit sequences represented as 32x32 RGB images, collected from Google Street View. The objective was to recognize the digit sequences in these images. The performance of each pretrained model was evaluated using multiple metrics on the SVHN test set.

4.2 Dataset Description

The SVHN dataset comprises a training set and a test set, containing digit sequences represented as 32x32 RGB images. To reduce computational complexity, the training set was subsampled by selecting every fourth image (considering only 25% of data).

4.3 Methodology

4.3.1 Data Preprocessing

Standard data preprocessing techniques were applied to the SVHN dataset, including resizing, random rotation, random cropping, horizontal flipping, and normalization. These transformations were crucial for preparing the input images for training the CNN models effectively.

4.3.2 Pretrained Models

A variety of pretrained CNN models were selected for evaluation, spanning different architectures such as LeNet-5, VGG-16, ResNet-18, ResNet-50, ResNet-101, WideResNet-50-2, and MNASNet-1.0. These models were pretrained on large-scale image datasets like ImageNet and were fine-tuned on the SVHN dataset.

4.3.3 Training

The pretrained models were fine-tuned on the SVHN dataset using the Adam optimizer with a learning rate of 0.001. Data augmentation techniques were employed during training to improve generalization performance. The training process lasted for 10 epochs.

4.4 Evaluation Metrics

The performance of each model on the SVHN test set was evaluated using the following metrics:

- Test Accuracy: The percentage of correctly classified images.
- Precision: The ratio of true positive predictions to the total number of positive predictions.
- Recall: The ratio of true positive predictions to the total number of actual positives.
- F1 Score: The harmonic mean of precision and recall, providing a balance between the two metrics.

4.5 Results

The performance of each pretrained model on the SVHN test set is summarized in Table 1.

Table 1: Performance of Pretrained Models on SVHN Test Set

Model	**Test Accuracy**	**Precision**	**Recall**	**F1 Score**
LeNet-5	76.04%	0.7639	0.7604	0.7593
VGG-16	19.59%	0.0384	0.1959	0.0642
ResNet-18	87.21%	0.8754	0.8721	0.8713
ResNet-50	87.86%	0.8834	0.8786	0.8789
ResNet-101	80.46%	0.8058	0.8046	0.8029
WideResNet-50-2	35.87%	0.3040	0.3587	0.2870
MNASNet-1.0	32.88%	0.7055	0.3288	0.3109

4.6 Conclusion

In conclusion, the evaluation results highlight significant variations in the performance of pretrained models on the SVHN dataset. Models like ResNet-18 and ResNet-50 exhibited high accuracy and demonstrated robust precision, recall, and F1 scores, indicating their effectiveness in digit recognition tasks because ResNet-50 is well-suited for the SVHN dataset due to its depth, skip connections, pretrained weights on ImageNet, global average pooling, parameter efficiency, and robustness to overfitting. These architectural characteristics enable ResNet-50 to effectively learn and generalize from digit sequences in the SVHN dataset, leading to high accuracy and robust performance in digit recognition tasks. Conversely, simpler models like LeNet-5 and VGG-16 showed inferior performance, emphasizing the importance of model complexity in achieving satisfactory results on specific datasets. WideResNet-50-2 and MNASNet-1.0 exhibited relatively lower accuracy and suboptimal precision-recall trade-offs, suggesting potential limitations of their architectures for the SVHN dataset.