Classification of Family Types of Proteins

Name:	**Anshuman Dangwal**
Registration No./Roll No.:	21044
Institute/University Name:	IISER Bhopal
Program/Stream:	Data Science and Engineering

1 Question 1

In logistic regression, the aim is to find the optimal decision boundary that best separates the classes in the feature space. The Cross Entropy loss function directly optimizes this decision boundary by maximizing the likelihood of observing the true labels given the model's parameters. In contrast, using MSE might lead to suboptimal results because it does not directly consider the probabilistic nature of the problem.
Moreover, in logistic regression, the output is a probability estimate between 0 and 1. Cross Entropy loss aligns better with this probabilistic interpretation, as it directly measures the discrepancy between the predicted probabilities and the true labels.

Using Cross Entropy loss in logistic regression ensures that the model aims to provide a single best answer for each instance, as it penalizes deviations from the true labels according to the logarithmic difference between predicted and actual probabilities. This is important because in classification tasks, especially when the model is deployed in real-world applications, we often need clear and confident predictions rather than vague or ambiguous ones.

Regarding the training process, using Cross Entropy loss typically results in faster convergence and better generalization performance compared to MSE, especially in scenarios where the classes are imbalanced or when the decision boundary is non-linear. This is because Cross Entropy loss guides the model to make more confident predictions, leading to a more decisive and effective learning process.

2 Question 2

For a given input X, if all the activation functions in the MLP are linear then the entire network becomes a linear function of the input X i.e.

$$\hat{y_i} = w_i x + b_i$$

Now the loss functions for cross entropy and MSE would be:

$$L_{CE} = -\sum_{i=1}^{2} y_i log(\hat{y_i})$$

Normally, the y_i factor only is 1 when is the index of the correct class. Therefore, with each class, the function is just:

$$f(x) = -log(x)$$

Now, to prove this one is convex, we have to compute the second derivative:

$$\frac{\partial L}{\partial x} = -\frac{1}{x} \Rightarrow \frac{\partial^2 L}{\partial x^2} = \frac{1}{x^2} > 0$$

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

The gradient of the MSE loss function with respect to the parameters is:

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

Since $(y_i - \hat{y}_i)^2$ is a convex function of $\hat{y}_i$, and the sum of convex functions is convex, the MSE loss function is convex.

The answer is option (c) both

3 Question 3

1. **Model Architecture**: The `FeedForwardNN` class defines a feedforward neural network with dense layers only. It takes input size, list of hidden layer sizes, output size, and list of activation functions as arguments. It creates a sequential model with linear layers and specified activation functions.

2. **Preprocessing**: For preprocessing input images, we use `torchvision.transforms.Compose` to apply a series of transformations. Here, we convert images to tensors and normalize them using mean and standard deviation.

3. **Dataset**: We load the MNIST dataset using `torchvision.datasets.MNIST` and apply the defined transformations.

4. **Data Loaders**: We create data loaders for training and testing datasets using `torch.utils.data.DataLoad`

5. **Training Loop**: We train the model using a simple training loop. We iterate through batches of data, perform forward pass, calculate loss, backward pass, and update weights using Adam optimizer.

6. **Evaluation**: We evaluate the trained model on the test dataset and print the accuracy.

For hyperparameter tuning strategies, one can consider techniques like grid search, random search, or Bayesian optimization to search through the hyperparameter space efficiently. Additionally, techniques like learning rate schedules, weight decay, dropout, and batch normalization can be employed for better performance and generalization of the model.

4 Question 4

For SVHN dataset, models like VGG, ResNet, and AlexNet, which have deep architectures with a large number of parameters and learn hierarchical features, are well-suited. These models have been pretrained on ImageNet, a large dataset with similar characteristics to SVHN, and hence they capture useful feature representations for this dataset. Additionally, these models have shown strong performance on various image classification tasks. Specifically, ResNet variants are known for their effectiveness in handling vanishing gradient problem and facilitating deeper architectures. Hence, ResNet-50 or ResNet-101 might be particularly well-suited for SVHN dataset due to their deeper architectures.