Deep Learning Assignment Report- **Vishesh Kumar(2221002)**

Q1. 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.

Ans. In the context of logistic regression, the most appropriate loss function is the Cross-Entropy loss (also known as Log Loss) rather than the Mean Squared Error (MSE). This choice is pivotal for the model's convergence and performance, primarily due to the nature of logistic regression as a classification problem solver, specifically for binary classification tasks.

1. **Probability Interpretation:** Logistic regression models the probability that a given input belongs to a particular class. Cross Entropy loss is a natural fit for probability distributions. It measures the distance between the model's predicted probability distribution and the actual distribution (the true labels). Since logistic regression outputs probabilities, using a loss function that operates directly on these probabilities is logical and efficient.

2. **Gradient Behavior:** The gradient of the Cross-Entropy loss function for the weights leads to a simpler and more stable optimization problem. For logistic regression, the gradients of the Cross-Entropy loss have the form that encourages faster convergence towards the minimum because the gradient is proportional to the error term (the difference between the predicted probability and the actual label). This characteristic helps avoid learning rate issues that can arise with MSE, where the gradient can become very small when the model is wrong, slowing down the learning process (a problem known as vanishing gradients).

3. **Convexity:** In logistic regression, Cross Entropy loss produces a convex loss surface, ensuring a single global minimum. This is crucial for gradient descent optimization methods to effectively find the best model parameters without getting trapped in local minima. MSE, on the other hand, does not guarantee a convex loss surface for logistic regression, making it possible for optimization to converge to suboptimal solutions.

Q2. 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

Ans. d An optimization problem is convex if its objective function is convex. This means for any two points within the domain, the function's value at any point on the straight line connecting these two points is less than or equal to the line segment connecting the function values at these two points. Mathematically, a function f is convex if for all x_1, x_2in the domain and $\theta \in [0, 1]$,

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta)f(x_2)$$

Cross Entropy (CE) Loss: CE is generally not convex in the model's parameters for neural networks, even if the activation functions are linear. The reason is that the output of a neural network with linear activations (and without any regularization term that could potentially introduce convexity) can be represented as a linear combination of inputs weighted by the parameters of the network. The composition of a linear function with the CE loss function does not result in a convex function concerning the network parameters.

Mean Squared Error (MSE) Loss: Similarly, MSE loss is not convex in the parameters of a neural network for binary classification tasks, regardless of the linear activation functions. While MSE is a convex function in terms of prediction errors, the parameter space of a neural network introduces non-convexity due to the nature of the compositions and multiplications of

parameters in the network, especially with multiple layers involved.

Q3. 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.
Ans. Implementation details of Dense Neural Network:

- Number of hidden layers = 3
- Number of hidden neuron = [32,64,128]
- Activation function = Relu, Tanh, Sigmoid

Preprocess the input images: I have used the MNIST dataset, which consists of 28×28 pixel grayscale images of handwritten digits from 0 to 9. For preprocessing I have converted each sample represented as a 3D array with dimension $28 \times 28 \times 1$

Activation Function	**Neurons per Layer**	**Mean Test Score**
ReLU	32	0.92
	64	0.93
	128	0.94
	256	0.95
Tanh	32	0.92
	64	0.93
	128	0.93
	256	0.93
Sigmoid	32	0.90
	64	0.91
	128	0.91
	256	0.92

Table 1: Results for Different Activation Functions

Q4. 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. (You can use a subset of dataset (25 %) in case you do not have enough compute.)
Ans. The SVHN dataset is composed of house number images sourced from Google Street View, where each image includes digits that need to be classified accurately. Regarding preprocessing, the images undergo resizing to dimensions of 32×32, alongside augmentations such as random rotations, crops, and horizontal flips. The models evaluated on this dataset include (1) LeNet-5, (2) VGG-16, (3) ResNet-18, (4) ResNet-50, and (5) ResNet-101.

Compared to LeNet-5, the latter models are significantly more intricate and deep, making them adept at managing complex image classification tasks. However, their effectiveness is reduced for datasets with small images like SVHN. The more profound architecture of these models may lead to overfitting or challenges in capturing critical features from the images, which can negatively affect performance. **For results of each model, please see the code**.