Q1. Interpretation of Loss Functions

Interpretation:

1. **Cross-Entropy**: Measures the disparity between the predicted probability distribution and the actual distribution. Lower Cross-Entropy signifies superior model performance as the predicted probabilities closely match the true labels.
2. **Mean Squared Error (MSE)**: Computes the squared discrepancy between predicted continuous values and actual target values. Ideal for regression tasks where prediction involves continuous values but lacks a direct translation to the model's confidence in class prediction for classification tasks with discrete class labels.

Output and Loss Landscape:

1. **Logistic Regression Output**: Ranges between 0 and 1, representing class probabilities.
2. **Cross-Entropy Loss**: Tailored for the logistic regression output range, penalizing significant deviations from true probabilities, particularly at extremes (0 or 1). Encourages the model to converge towards distinct class boundaries, aiming for a singular optimal classification.
3. **Impact on Training**:

- **Cross-Entropy Loss**: Drives model adjustment to minimize discrepancies between predicted probabilities and true labels. Promotes learning of effective class boundaries, facilitating clear classification decisions.
- **Mean Squared Error (MSE)**: Inappropriate for logistic regression output, as it would not penalize small deviations from true labels, potentially resulting in ambiguous predictions around 0.5.

Conclusion:

Cross-Entropy loss is indispensable for logistic regression due to its alignment with model output (probabilities) and training objective (clear classification). By penalizing significant deviations from true labels, particularly at extreme probabilities, Cross-Entropy loss facilitates convergence towards a singular optimal classification, thus enhancing the model's ability to provide definitive predictions.

Q2. Correct Loss Function in Binary Classification

The right response is loss of cross-entropy. Because Cross-Entropy loss is convex, convex optimization is guaranteed. For binary classification tasks using deep neural networks, the empirical evidence of convexity is commonly applied and proven to be effective, despite its potential complexity. Mean Squared Error (MSE) loss, on the other hand, lacks convexity and frequently results in several local minima. MSE could not be taking full advantage of the network's potential, particularly when using linear activation functions. Because CE loss guarantees a convex optimization issue, deep neural network-based binary classification tasks favor it over other options. This justifies the incorrectness of options (c) and (d).

Q3. Code Explanation for MNIST Digit Recognition
Data Loading and Preprocessing:

The dataset is divided into training and testing sets, denoted as $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}$.

Pixel values of images are normalized to the range [0, 1] by dividing by 255.0.
The code starts by loading the MNIST dataset using `datasets.mnist.load`$_{d}ata$.`The code starts by loading the MNIST dataset`

Model Architecture:

The neural network model is built using the Sequential API.
It comprises a flattening layer followed by three fully connected Dense layers.
The first dense layer has 128 units with ReLU activation, followed by 64 units with sigmoid activation,
and then 32 units with hyperbolic tangent (tanh) activation.
The final dense layer with 10 units employs softmax activation for multi-class classification.

Compilation, Training, and Evaluation:

For compilation, the model is optimized using the SGD optimizer
and the loss function used is sparse categorical cross-entropy. Accuracy is chosen as the evaluation metric.
The model undergoes training using `model.fit` on the training data for 5 epochs,
with evaluation on the validation data provided by `validation_data`.
After training, the model is evaluated on the test set using `model.evaluate`,
and the test accuracy is printed out.
Overall, this code demonstrates a basic neural network setup for the MNIST dataset, showcasing common practices
in data loading, model building, and evaluation using TensorFlow and Keras.

Q4.

Code Explanation

This code trains and evaluates several pre-trained models (AlexNet, VGG-11, and ResNet-18) on a subset of the Street View House Numbers (SVHN) dataset for digit recognition. Here's a breakdown:

Data Loading and Preprocessing:

- Defines transformations to resize images to 64x64 pixels and normalize pixel values.
- Loads the SVHN dataset, including both training and test sets, and creates subsets containing 25

Model Initialization:

- Initializes three pre-trained models: AlexNet, VGG-11, and ResNet-18.
- Adjusts the output layer of each model to have 10 units for digit classification.

Training:

- Trains each model for one epoch (due to computational constraints).
- Utilizes cross-entropy loss and ADAM) optimizer with a higher learning rate for quicker convergence.
- Outputs the average loss per epoch during training.

Evaluation:

- Evaluates each trained model on the test set.
- Computes the accuracy of each model on the test set and displays the results.

The code exemplifies the process of fine-tuning pre-trained models for a specific task (digit recognition) on a subset of the SVHN dataset. It underscores the significance of transfer learning, enabling the utilization of pre-trained models to achieve reasonable accuracy even with limited training data.