**Name:- KARAN TARIYAL Roll Number:- M22AI563 Subject:- CSL7020**

Machine Learning - I

Fractal 3 Assignment

**INSTRUCTOR: Dr. Anand Mishra TOPIC: Fractal 3 Assignment**

**Q1**

1. To determine the number of steps it takes for the perceptron learning algorithm to converge, we can use the following formula:

𝑛 <= 𝑅^2 / 𝛾^2

where 𝑛 is the number of steps, 𝑅 is the maximum length of the feature vectors, and 𝛾 is the minimum margin. In this case, 𝑅 = sqrt(2), as the maximum length of feature vectors is sqrt(x1^2 + x2^2). We can set 𝛾 = 1, as we want a decision boundary that correctly classifies all the training samples.

Plugging in the values, we get:

𝑛 <= (sqrt(2))^2 / (1)^2 𝑛 <= 2

This means that the perceptron learning algorithm will converge in at most 2 steps.

1. We can update the weight vector using the perceptron learning algorithm as follows:

Initialize the weight vector: w=[1,1] Initialize the bias term: b=0 Set the learning rate: 𝜂=1

For each sample (xi, yi) in the training data: if yi \* (w.T \* xi + b) <= 0: w = w + 𝜂 \* yi \* xi b = b + 𝜂 \* yi

Using this algorithm, we can update the weight vector for each misclassified sample until we obtain a decision boundary that correctly classifies all the training samples.

**Step 1:** For the first sample (1, 1, +1), we have: y = +1 w.T \* x + b = 2 y \* (w.T \* x + b) = +2 > 0 No update needed.

**Step 2:** For the second sample (-1, -1, -1), we have: y = -1 w.T \* x + b = 0 y \* (w.T \* x + b) = -0 > 0 Update needed: w = w + 𝜂 \* y \* x = [1,1] + 1 \* [-1,-1] = [0,0] b = b + 𝜂 \* y = 0 + 1 \* (-1) = -1

**Step 3:** For the third sample (0, 0.5, -1), we have: y = -1 w.T \* x + b = -0.5 y \* (w.T \* x + b) = 0.5 > 0 No update needed.

**Step 4:** For the fourth sample (0.1, 0.5, -1), we have: y = -1 w.T \* x + b = -0.4 y \* (w.T \* x + b) = 0.4 > 0 No update needed.

**Step 5:** For the fifth sample (0.2, 0.2, +1), we have: y = +1 w.T \* x + b = 0.2 y \* (w.T \* x + b) = 0.2 > 0 No update needed.

**Step 6:** For the sixth sample (0.9, 0.5, +1), we have: y = +1 w.T \* x + b = 1.4 y \* (w.T \* x + b) = 0.7 > 0 No update needed.

The final weight vector is w = [0,0] and the bias term is b = -1. The decision boundary is w.T \* x + b = 0, which

**Q2**

Firstly, you need to preprocess the dataset. This includes normalizing the data, which means scaling the pixel values of the images to a range between 0 and 1. You can use the **MinMaxScaler** from the **sklearn** library for this purpose. Then, you need to split the data into training, validation, and test sets. You can use the **train\_test\_split** function from the **sklearn** library for this purpose.

Next, you can create a neural network model using a simple architecture. A simple neural network for image classification can have an input layer, one or more hidden layers, and an output layer. The number of nodes in the input layer will be equal to the number of features in your dataset, which in this case is the number of pixels in each image. The number of nodes in the output layer will be equal to the number of classes you want to classify, which in this case is 10.

You can experiment with different numbers of hidden layers and nodes in each layer to optimize the performance of your model. You can use the **Dense** layer from the **keras** library to create the layers of your neural network.

Finally, you need to compile and train your model using the training data. You can use the **compile** and **fit** methods of the **keras** library for this purpose. Once your model is trained, you can evaluate its performance on the validation and test sets.

Some **observations** you can make on your implementation include the accuracy and loss values of your model on the training, validation, and test sets. You can also plot the learning curves of your model to see how it is learning over time. Additionally, you can experiment with different hyperparameters, such as the learning rate and batch size, to further optimize your model's performance.

I hope this guidance helps you in implementing a neural network for Gurmukhi Handwritten Digit Classification.

**Q3**

**Task 1**: To start with, you need to download the dataset from the given link and split it into training and validation sets. You can use the **train\_test\_split** function from the **sklearn** library to split the dataset in an appropriate ratio. You can also use the CSV file to get the labels for the images.

**Task 2**: For implementing a two-layer CNN, you can use the **Conv2D** and **MaxPooling2D** layers from the **keras** library. The first layer will be a convolutional layer, which will extract features from the input image. The second layer will be a max pooling layer, which will downsample the output of the convolutional layer. You can then add a flatten layer to convert the output of the second layer into a 1D feature vector, followed by a dense layer for classification.

You can compile and train your model using the **compile** and **fit** methods of the **keras** library. Once your model is trained, you can evaluate its performance on the validation set and plot the obtained loss using the **matplotlib** library.

**Task 3**: For finetuning a pre-trained network, you can use the **AlexNet** architecture as a starting point. You can download the pre-trained weights of the **AlexNet** model using the **keras.applications** module. You can then freeze the layers of the pre-trained model up to a certain point and add new layers for classification.

You can compile and train your finetuned model using the same approach as in Task 2. Once your model is trained, you can evaluate its performance on the validation set.

**Observations**: You can observe the accuracy and loss values of your models on the training and validation sets. You can also compare the performance of your two-layer CNN model and the finetuned **AlexNet** model. You can experiment with different hyperparameters, such as the learning rate and batch size, to optimize the performance of your models.

I hope this guidance helps you in implementing a CNN-based classification architecture for chart image classification.