Single Perceptron.

The single perceptron learning rule, also known as the perceptron algorithm, is a simple supervised learning algorithm used for binary classification tasks. Here's a brief explanation:

Perceptron Model:

A perceptron is a basic computational unit in neural networks. It takes several input signals, each multiplied by a corresponding weight, sums them up, and applies an activation function to produce an output.

Binary Classification:

The perceptron is used for binary classification, where it predicts whether an input belongs to one class (e.g., +1) or another class (e.g., -1).

Learning Rule:

The goal of the perceptron learning rule is to adjust the weights of the perceptron based on training data to minimize classification errors.

Training Process:

Initialize the weights (w) and bias (b) of the perceptron to small random values.

For each training example (x, y) where x is the input vector and y is the true class label:

Compute the predicted class label $\hat{y}$ using the current weights and bias: $\hat{y} = \text{sign}(wi* xi + b)$, where w is weight, sign is a function that outputs +1 or -1 depending on the sign of its input.

Update the weights and bias if the prediction $\hat{y}$ is incorrect:

If $\hat{y} \neq y$, update weights:

L.R*(ACT - PREDICTED) * INPUT WITH RESPEST TO WEIGHT

Update bias: b = BIAS (ACTUAL - PREDICTED)

Repeat the above steps for a number of epochs or until convergence (when the model stops making errors or the error rate falls below a threshold).

Decision Boundary:

The perceptron algorithm learns a linear decision boundary that separates the two classes in the input space.

Limitations:

The perceptron learning rule works only for linearly separable data. If the data is not linearly separable, the algorithm may not converge to a solution.

In summary, the single perceptron learning rule is a foundational algorithm for binary classification, adjusting weights based on prediction errors to learn a linear decision boundary between classes.

Task 1 and 2:

Write a Python program to create a single perceptron and train this perceptron on AND and OR problems.

Convolutional neural network

Convolutional Layers: These are the building blocks of CNNs. A convolutional layer applies learnable filters (also called kernels) to small regions of the input image. Each filter extracts specific features (such as edges, textures) from the input image by performing a convolution operation.

Pooling Layers: After each convolutional layer, a pooling layer is typically added. Pooling layers downsample the feature maps generated by the convolutional layers. Common pooling operations include max pooling (selecting the maximum value in a region) or average pooling (taking the average value in a region). Pooling helps in reducing the spatial dimensions of the feature maps and makes the learned features more invariant to small transformations.

Activation Functions: Non-linear activation functions (like ReLU - Rectified Linear Unit) are applied after each convolutional and pooling layer to introduce non-linearity into the network and enable it to learn complex patterns.

Fully Connected Layers: Towards the end of the CNN architecture, fully connected layers are added. These layers take the high-level features extracted by the convolutional layers and learn to classify the input into various classes based on these features. The final fully connected layer usually employs a softmax activation function for classification tasks to output the probabilities of different classes.

Training: CNNs are trained using a supervised learning approach with labeled training data. They optimize their internal parameters (the weights and biases of the filters) using backpropagation and gradient descent to minimize a loss function, typically a measure of the difference between the predicted outputs and the true labels.

CNNs are widely used in tasks such as image classification, object detection, facial recognition, and more. Their architecture, which mimics the visual cortex's organization in animals, has proven to be highly effective in extracting and learning hierarchical representations from visual data.

Task 2

Apply DenseNet on the CIFAR dataset, use Random Forest as a classifier, and plot graphs with different numbers of trees.

Freeze all layers except the last convolutional block of VGG16, fine-tune the model on the CIFAR dataset using different dense layers, and print the test accuracy.