The assignment focuses on designing and implementing a custom activation function for a neural network. The task requires creating a neural network that adapts its activation function (AF) dynamically, rather than using pre-existing activation functions like ReLU, Sigmoid, etc.

Problem Summary

- You need to design a flexible activation function of the form:

  $g(x)=k0+k1 \cdot x$

  where the parameters k0 and k1 are learned during training.
- You should avoid using traditional brute-force grid search methods and instead focus on the neural network's ability to learn these parameters.

Suggested Approach

1. **Feedforward Network**:
   - Create a neural network with 1 or 2 hidden layers.
   - The activation function for each layer is $g(x)=k0+k1 \cdot x$, where k0 and k1 are learnable parameters.
   - You'll use categorical cross-entropy as the loss function, appropriate for classification tasks.
2. **Custom Activation Function**:
   - Define a new activation function that incorporates the learned parameters k0 and k1.
   - During backpropagation, compute the gradients with respect to these parameters and update them using gradient descent.
3. **Training**:
   - The neural network will be trained using datasets such as Bank-Note, Iris, Wisconsin Breast Cancer, or MNIST.
   - Track the model’s accuracy, loss, and F1-Score during training.

Explanation and full code is in .ipynb file

1. **Custom Activation Class**:
   - A new layer, `CustomActivation`, is created where two trainable parameters, `k0` and `k1`, are initialized and used to define the activation function $g(x)=k0+k1 \cdot x$.
2. **Model Structure**:
   - The model is built using two hidden layers, each with the custom activation function.
   - The output layer uses the softmax activation function, suitable for multi-class classification tasks like MNIST.
3. **Training**:
   - The model is compiled with Adam optimizer and categorical cross-entropy loss function, and it's trained for 10 epochs.
4. **Evaluation**:

- After training, the model is evaluated on the test dataset, and the accuracy is printed.

Key Points:

- **Trainable Activation Parameters**: The custom activation function allows the model to learn optimal values for k0 and k1 during training.
- **Backpropagation**: The TensorFlow library automatically computes the gradients for the parameters k0 and k1, updating them during training using the Adam optimizer.

Further Improvements:

- Experiment with different datasets like Iris or Wisconsin Breast Cancer to see how well the model generalizes to different data.
- Plot the loss vs. epochs and accuracy metrics to analyze the model's learning behavior.

This approach satisfies the requirement of using a custom activation function and allows the neural network to learn the best parameters for the activation function automatically.

1. **`matplotlib`**:
    - `matplotlib.pyplot` is used to create plots and visualize the training process.
2. **Training and Validation Loss/Accuracy**:
    - The model's history object (`history`) records the loss and accuracy for both training and validation over each epoch.
    - Two line plots are created: one for loss and one for accuracy, to visualize how the model improves over time.
3. **Bar Plot**:
    - After training, a bar plot is generated that compares the final epoch's training accuracy with validation accuracy.

Output:

1. **Line Plots**:
    - **Training and Validation Loss**: Shows how the loss decreases over time for both the training and validation datasets.
    - **Training and Validation Accuracy**: Displays how the accuracy improves over the epochs.
2. **Bar Plot**:
    - Displays a side-by-side comparison of training accuracy and validation accuracy for the final epoch.

Visualization

These graphs will help you observe the model’s performance over time and compare how well it generalizes by evaluating the gap between training and validation metrics.