Technical Report: Learning Activations in Neural Networks

Introduction:-

Through this test , the creation of an activation function , known as Ada - Act is being explored. It adapts the dataset by learning the coefficients and making the results. We all know that the choice of activation functions severely affects the results of the ANNs. The aim of this project is to design a neural network that can automatically learn the best activation function form using parameters learned through training. avoiding brute-force selection of pre-existing functions. The function takes the form:

g(x)= ko+k1.(x)

where the coefficients Ko and K1 are learned during training. This report details the methodology, implementation, and performance of the model trained on the Wisconsin Breast Cancer dataset.

Data Preprocessing:-

The dataset used for this implementation is the Wisconsin Breast Cancer Dataset. After loading the dataset, missing values were handled, and categorical target labels were encoded using LabelEncoder. The dataset was split into training and testing sets, and the features were standardized to improve the learning rate and performance of the model.

Code:-

```
# Load dataset
data = pd.read_csv('/content/wisc_bc_data.csv')
data = data.dropna()  # Ensure no NaN values

# Encode target column
label_encoder = LabelEncoder()
data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

# Split data
X = data.drop('diagnosis', axis=1).values
y = data['diagnosis'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Ada-Act Activation Function

The core of this implementation is the custom activation function, **Ada-Act**, which allows the neural network to learn its optimal activation function during training.The function g(x)=k0+k1∗xg(x) = k_0 + k_1 * xg(x)=k0+k1∗x adds flexibility to the model, as it adjusts itself dynamically based on the dataset's characteristics. The coefficients k0k_0k0 and k1k_1k1 are learned during back-propagation, just like other network parameters, making this an adaptive approach.

Code:-

```
class AdaAct(nn.Module):
def __init__(self):
super(AdaAct, self).__init__()
# Learnable parameters k0 and k1
self.k0 =
nn.Parameter(torch.randn(1))
self.k1 =
nn.Parameter(torch.randn(1))

def forward(self, x):
return self.k0 + self.k1 * x
```

Neural Network Architecture

The neural network model consists of:

- Input layer: 31 nodes corresponding to the 31 features.
- Hidden layer: 1 hidden layer with 30 neurons.
- Output layer: 2 nodes for binary classification with a softmax activation.

The Ada-Act function is applied after the first layer, allowing the network to learn the best activation function during training.

Ada-Act(x)= k0+k1.(x)

The parameters K0 and K1 are initialized randomly and updated via backpropagation along with the weights of the network. The categorical cross-entropy loss function was used for optimization with the Adam optimizer.

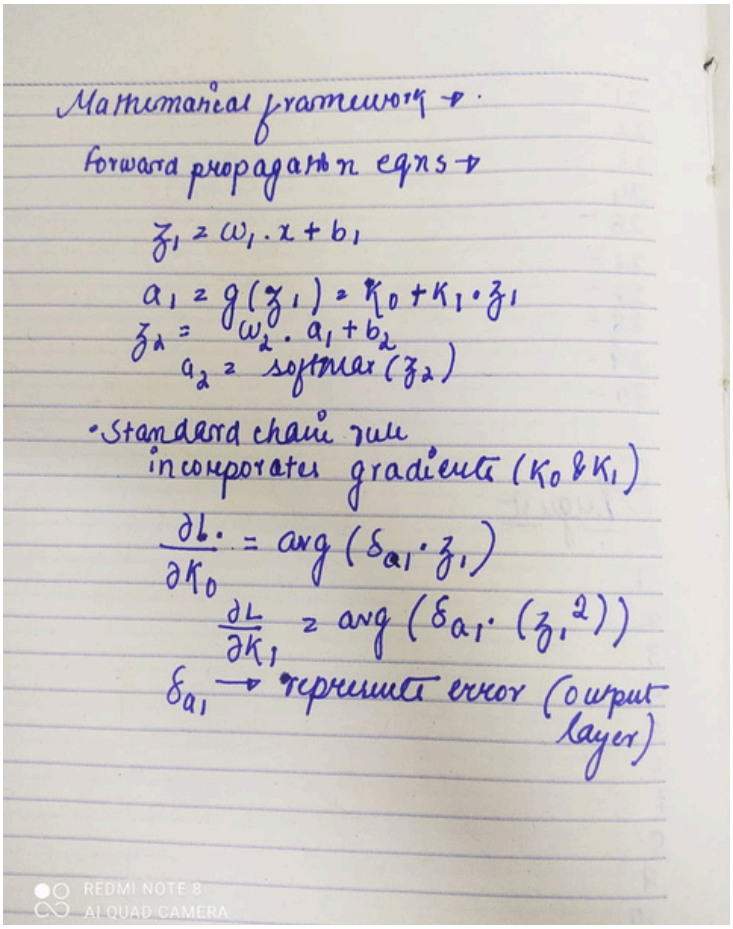
```
# Load the trained model
input_size = 31 # Use the same size as during training, e.g., 31 if that was used
hidden_size = 30
output_size = 2
model = NeuralNet(input_size, hidden_size, output_size)
```

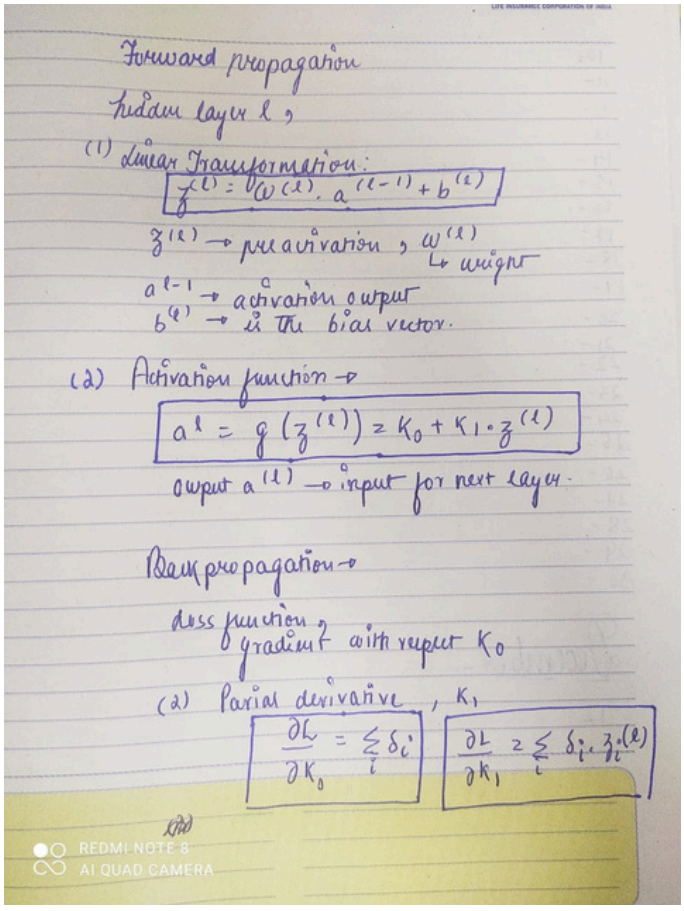
Mathematical Framework:-

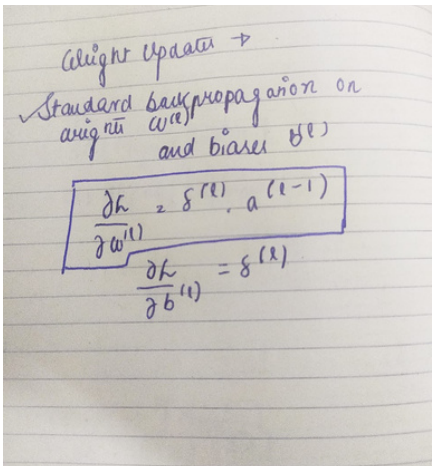
Unlike traditional fixed activation functions like ReLU, Sigmoid, or Tanh, Ada-Act learns these coefficients, allowing it to adapt to the data dynamically. This flexibility introduces additional non-linearity and ensures that the activation function is tailored to the specific task during training.

Forward Propagation

In a neural network, the forward propagation steps involve computing the weighted sum of inputs, passing them through an activation function, and then applying the activation output to the subsequent layer.







Model Implementation:-

- Hidden Size: 30 neurons
- Activation Function: Ada-Act
- Optimizer: Adam
- Loss Function: Categorical Cross-Entropy
- Learning Rate: 0.001
- Epochs: 100

The model was implemented in **PyTorch**, and training was conducted for 100 epochs. During training, both the train and test losses were monitored.

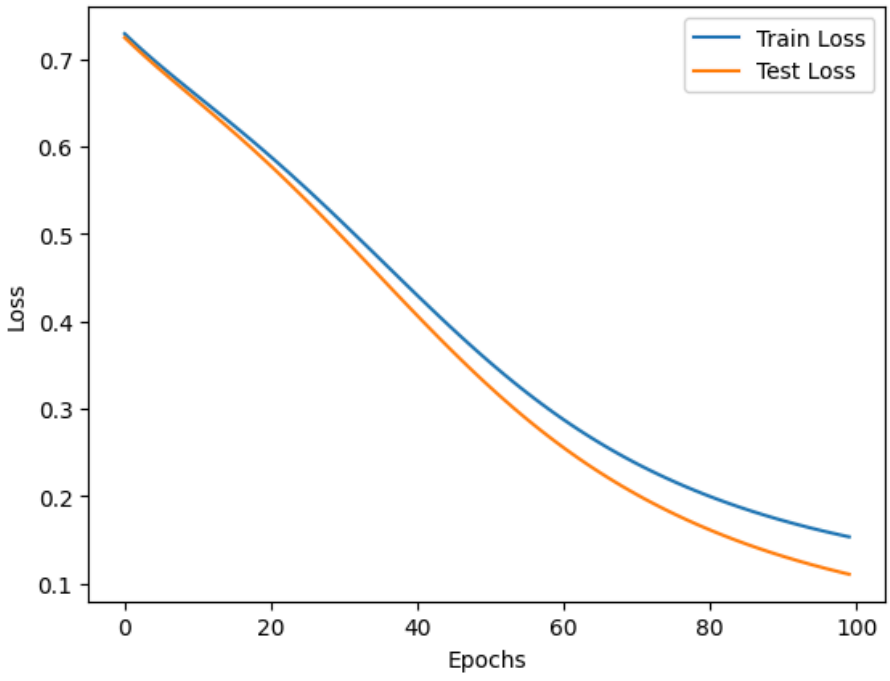
Results:-

The model was evaluated on test data after each epoch. The following metrics were recorded:

- **Accuracy**: 0.9825
- **F1-Score**: 0.9825
- **Precision**: 0.9833
- **Recall:** 0.9825

The **train and test loss** during training were as follows:

- Train Loss (Final Epoch): 0.1534
- Test Loss (Final Epoch):0.1106



Implementation of Working API :-

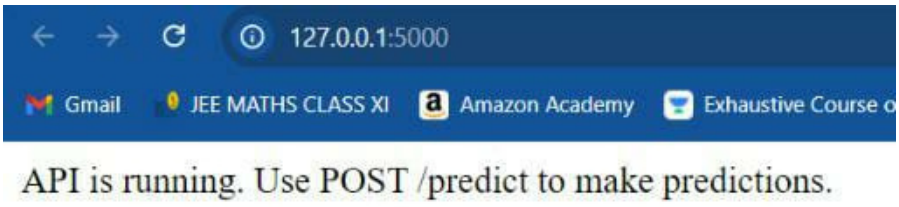
This Python Flask application provides an API that integrates with a custom neural network model, which uses a novel adaptive activation function (Ada-Act) to perform predictions. The model is a two-layer feedforward neural network, and the app allows for interaction with this model through HTTP requests.

Flask API Setup

The application uses the Flask web framework to create an API that can handle incoming requests. Here's an overview of how it works:

- Flask App Initialization: The Flask app is created with app = Flask(__name__). The app listens for HTTP requests and processes them based on the defined routes.
- Root Route (/): A simple route (GET /) is defined that returns a message confirming the API is running. This is a useful way to check the health of the API.

```
@app.route('/', methods=['GET'])
def home():
    return "API is running. Use POST /predict to make predictions."
```

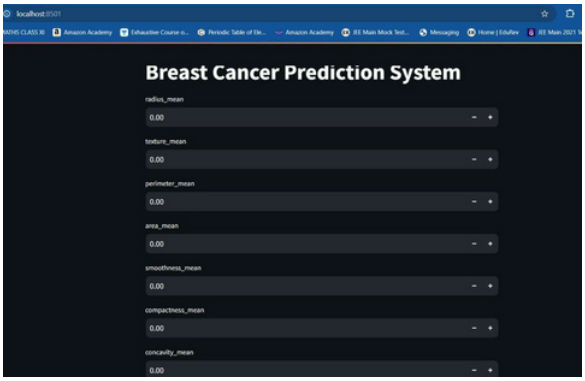


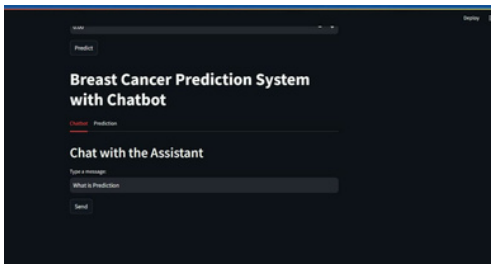
The API serves as the backend for a prediction app and a chatbot:

- **Prediction App:** The app collects user inputs or features and sends them to the /predict endpoint.
- **Chatbot Integration**: A chatbot can interact with this API by sending user queries (in the form of input data) to the Flask API and receiving predictions or responses in real-time.

Summary of the API Functionality:-

- Custom Neural Network: Implements a two-layer neural network with an adaptive activation function (Ada-Act) for flexible predictions.
- Model Deployment: Uses a Flask API to serve the trained model, allowing external systems (like a web app or chatbot) to interact with the model and get predictions.
- Prediction Endpoint: Accepts input data in JSON format, processes it, and returns predictions based on the trained model.





Github Link :-
https://github.com/Shrutakeerti/Assignment---27-09-2024
Prediction Web App - localhost:8501/
API testing link - http://127.0.0.1:5000/

Thank You