RA2311047010040

Index

மரம் வளர்ப்போம்
மண் வளம் காப்போம்

Superior® Note Books

Name : Vishva. S

Subject : DL

Std. : ..... Sec. : ..... Roll No. .....

School : .....

No.	Date	Title	Marks	Signature
1	24/7/25	Exploring the DL ~~platforms~~.		
2	31/7/25	Implement a classifier using open source dataset		
3	7/8/25	Study the classifier with respect to statistical parameters.		
4	14/8/25	Build a simple feed forward neural network to Recognize Handwritten characters		
5	22/8/25	Study of Activation functions & their role.		
6	9/9/25	Implement Gradient Descent & Backpropagation in Deep neural network.		

22/8/25 | Lab: 5 : Study of Activation functions & Their Role.

Aim:

To study different activation functions used in neural networks & analyze their role in learning & performance.

Objective:

→ To understand the purpose of activation functions in neural network.

→ To implement & visualize commonly used activation functions.

→ To compare their behavior & significance in training deep learning models.

→ To evaluate how different activation functions affect model performance.

Observation:

* Activation functions studied:

1) Sigmoid functions

$$f(x) = \frac{1}{1+e^{-x}}$$

* Range: (0,1)

* used for binary classification

2) Tanh function.

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

* Range: (-1,1)

* centered around zero, often better than Sigmoid.

3). ReLU (Rectified Linear Unit)

$f(x) = \max(0, x)$

* Range: $[0, \infty)$

* Most widely used. Helps solve vanishing gradient problem.

4) Leaky ReLU

$f(x) = x$ if $x \geq 0$, else $0.01x$

*Allows small gradient for negative inputs.

* Role of activation functions:

1) Introduce non-linearity in the network.

2) Allow neural networks to learn complex patterns.

3) Help control gradient flow during backpropagation.

Pseudocode:

BEGIN

Step 1: Select activation functions to study:

→ Sigmoid

→ Tanh

→ ReLU

→ Leaky ReLU

Step 2: Generate a set of input values in a given range (e.g.: -10 to +10)

Step 3: For each activation function:

Apply the function to the input values.

Store the corresponding outputs.

Draw the input-output graph.

Observe the curve & behaviour.

Step 4: Initialize a simple feed-forward neural network.

Step 5: For each activation function:

Replace the hidden layer activation with the chosen function.

Train the network on a sample dataset for fixed epochs

Record training accuracy and testing accuracy.

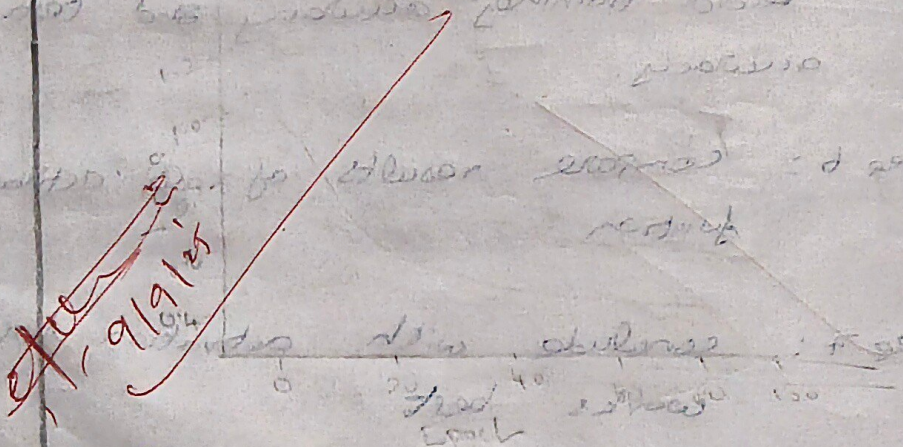
Step 6: Compare results of all activation function.

Step 7: Conclude with activation function works best

END.

Table

Activation function	Output range	Advantage	Use case
Sigmoid	(0,1)	Smooth, probabilistic output	Binary classification
Tanh	(-1,1)	Centred around 0, better than Sigmoid	Hidden layers (older networks)
ReLU	(0,∞)	Fast, reduce computation time	Hidden layers (most NN, ANN)
LeakyReLU	(-∞,∞)	Fixed ReLU (dying) issues	Deep hidden layers
Softmax	(0,1) sum = 1	give probability distribution	output layer for multi class



Result:

Studied different activation function their roles.

lab5_dlt.ipynb

File Edit View Insert Runtime Tools Help

Share Gemini

Commands + Code + Text Run all Connect

```
!pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.23.0+cu126)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.19.1)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.2)
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
```

```
iris = datasets.load_iris()
X = iris.data[:, :2]  # Take only first 2 features for easy plotting
y = iris.target
```

lab5_dlt.ipynb

File Edit View Insert Runtime Tools Help

Share Gemini

Commands + Code + Text Run all Connect

```
# Train/test split
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)
```

```
# Convert to torch tensors
X_train_t = torch.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.long)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_test_t = torch.tensor(y_test, dtype=torch.long)
```

```
class IrisNet(nn.Module):
    def __init__(self, activation='relu'):
        super(IrisNet, self).__init__()
        self.fc1 = nn.Linear(2, 16)
        self.fc2 = nn.Linear(16, 3)

        if activation == 'relu':
            self.activation = F.relu
        elif activation == 'tanh':
            self.activation = torch.tanh
        elif activation == 'sigmoid':
            self.activation = torch.sigmoid
        elif activation == 'leaky_relu':
            self.activation = F.leaky_relu
        else:
            raise ValueError("Unsupported activation")

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.fc2(x)  # No activation here because CrossEntropyLoss expects logits
        return x
```

lab5_dlt.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

Connect

Share Gemini V

```
X_train_t = torch.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.long)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_test_t = torch.tensor(y_test, dtype=torch.long)
```

```
class IrisNet(nn.Module):
    def __init__(self, activation='relu'):
        super(IrisNet, self).__init__()
        self.fc1 = nn.Linear(2, 16)
        self.fc2 = nn.Linear(16, 3)

        if activation == 'relu':
            self.activation = F.relu
        elif activation == 'tanh':
            self.activation = torch.tanh
        elif activation == 'sigmoid':
            self.activation = torch.sigmoid
        elif activation == 'leaky_relu':
            self.activation = F.leaky_relu
        else:
            raise ValueError("Unsupported activation")

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.fc2(x)  # No activation here because CrossEntropyLoss expects logits
        return x
```

```
def train_model(activation='relu', epochs=100, lr=0.01):
    model = IrisNet(activation)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()

    losses = []
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train_t)
        loss = criterion(outputs, y_train_t)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    return model, losses
```

lab5_dlt.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Share Gemini Connect

```
# 4. Train and plot loss for different activations
activations = ['relu', 'tanh', 'sigmoid', 'leaky_relu']
plt.figure(figsize=(10,6))

for act in activations:
    model, losses = train_model(activation=act)
    plt.plot(losses, label=act)

plt.title("Training Loss for Different Activations on Iris")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```