Experiments 5
1/9/2025

Activation Functions & Their roles

Aim: To learn about different activation functions and their roles.

Objective:

→ To evaluate and compare the performance of different activation functions

→ By training each model for a fixed number of epochs and measuring its test accuracy on unseen MNIST test data.

Pseudocode:

→ Load MNIST dataset

→ Define neural network architecture

→ Define list of activation function:
Sigmoid, Tanh, ReLu, LeakyReLu, SiLu

→ Train model for 5 epochs
- Forward pass
- Compute loss
- Backward pass
- Update weights

→ Compute accuracy

→ Display result.

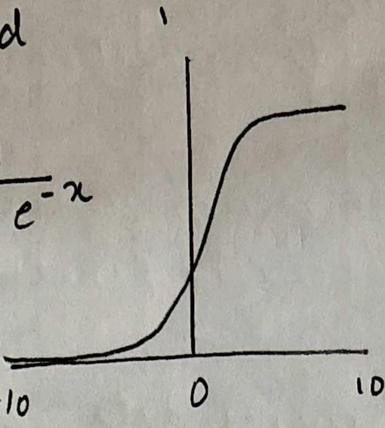
END

Observation

→ Overall high accuracy as MNIST digit recognition task is relatively simple and all activation functions handle it well.

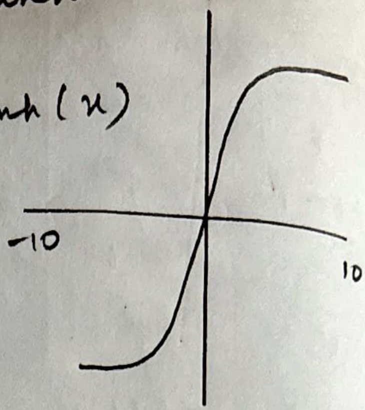
Sigmoid

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



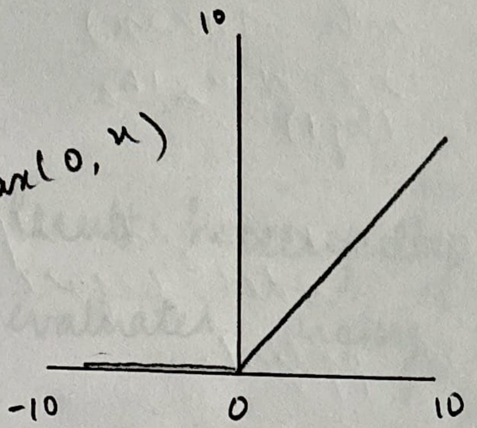
Tanh

$\tanh(x)$

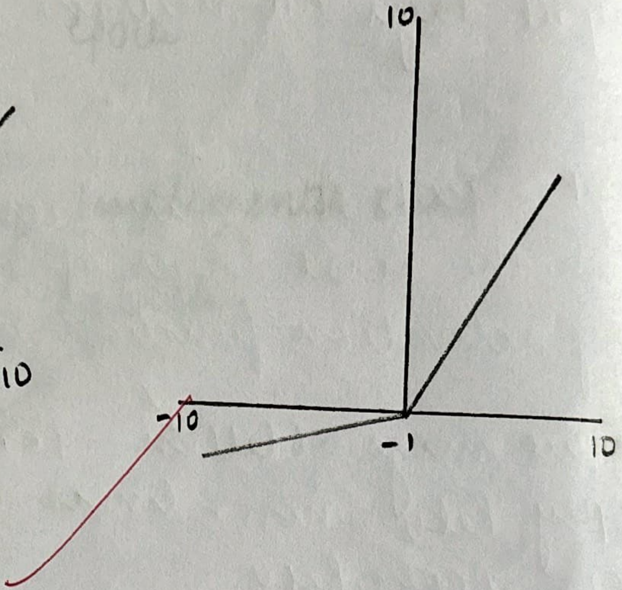


ReLU

$\max(0, x)$



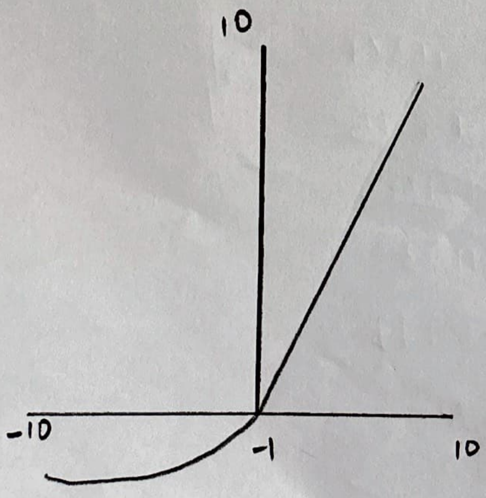
Leaky ReLU



Maxout

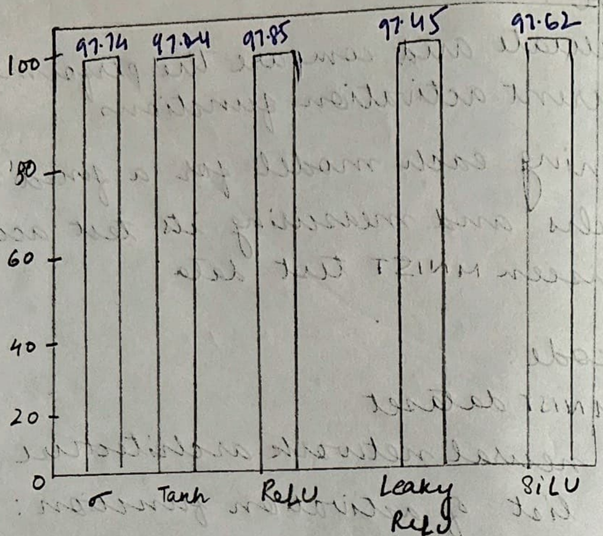
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU



$$\begin{cases} x & x \geq 0 \\ a(e^x - 1) & x < 0 \end{cases}$$

→ Among the activations, ReLU achieved the best accuracy because it avoids vanishing gradients and trains faster.



Sigmoid - 97.74%.

Tanh - 97.24%.

ReLU - 97.85%.

Leaky Relu : 97.45%.

SiLU : 97.62%.

Result: Trained the mode and learned about different activation functions successfully.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

x = torch.tensor([-2.0, -1.0, 0.0, 1.0, 2.0])

# Using module
sigmoid = nn.Sigmoid()
print(sigmoid(x))

# Using functional API
print(F.sigmoid(x))   # Deprecated, better to use torch.sigmoid
print(torch.sigmoid(x))
```

```
tensor([0.1192, 0.2689, 0.5000, 0.7311, 0.8808])
tensor([0.1192, 0.2689, 0.5000, 0.7311, 0.8808])
tensor([0.1192, 0.2689, 0.5000, 0.7311, 0.8808])
```

```python
tanh = nn.Tanh()
print(tanh(x))
print(torch.tanh(x))   # Functional
```

```
tensor([-0.9640, -0.7616,  0.0000,  0.7616,  0.9640])
tensor([-0.9640, -0.7616,  0.0000,  0.7616,  0.9640])
```

```python
relu = nn.ReLU()
print(relu(x))
print(F.relu(x))
```

In [ ]:
```
leaky_relu = nn.LeakyReLU(negative_slope=0.01)
print(leaky_relu(x))
print(F.leaky_relu(x, negative_slope=0.01))
```

```
tensor([-0.0200, -0.0100,  0.0000,  1.0000,  2.0000])
tensor([-0.0200, -0.0100,  0.0000,  1.0000,  2.0000])
```

In [ ]:
```
elu = nn.ELU()
print(elu(x))
print(F.elu(x))
```

```
tensor([-0.8647, -0.6321,  0.0000,  1.0000,  2.0000])
tensor([-0.8647, -0.6321,  0.0000,  1.0000,  2.0000])
```

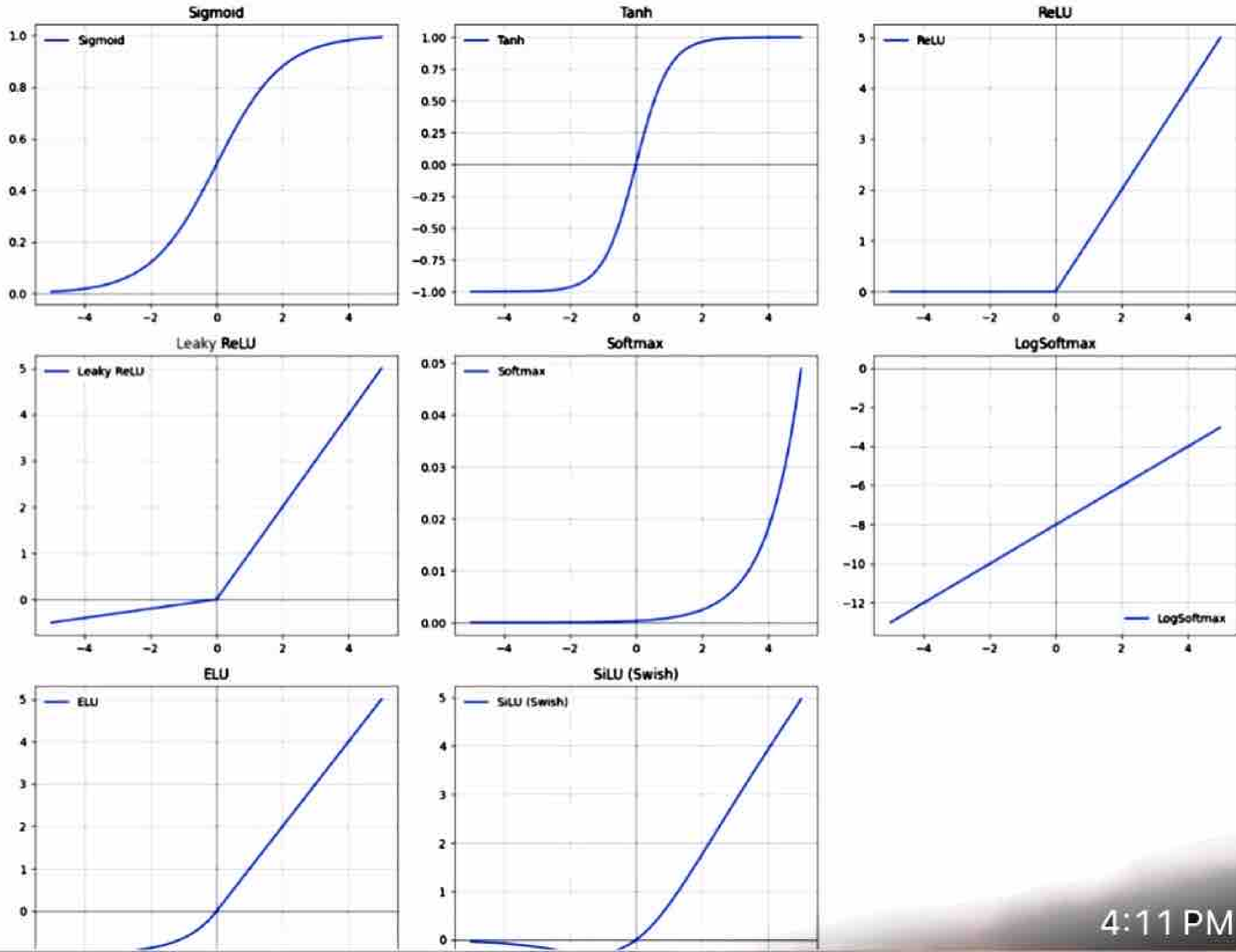
In [ ]:
```
y = torch.tensor([2.0, 1.0, 0.1])

softmax = nn.Softmax(dim=0)
print(softmax(y))
print(F.softmax(y, dim=0))
```

```
tensor([0.6590, 0.2424, 0.0986])
tensor([0.6590, 0.2424, 0.0986])
```

In [ ]:
```
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt

# Input values
x = torch.linspace(-5, 5, 200)
```

```python
# Activation functions
activations = {
    "Sigmoid": torch.sigmoid(x),
    "Tanh": torch.tanh(x),
    "ReLU": F.relu(x),
    "Leaky ReLU": F.leaky_relu(x, negative_slope=0.1),
    "Softmax": F.softmax(x, dim=0),
    "LogSoftmax": F.log_softmax(x, dim=0),
    "ELU": F.elu(x, alpha=1.0),
    "SiLU (Swish)": F.silu(x)
}

# Plotting
plt.figure(figsize=(15, 12))

for i, (name, y) in enumerate(activations.items(), 1):
    plt.subplot(3, 3, i)
    plt.plot(x.numpy(), y.detach().numpy(), label=name, color="blue")
    plt.title(name)
    plt.axhline(0, color="black", linewidth=0.5)
    plt.axvline(0, color="black", linewidth=0.5)
    plt.grid(True)
    plt.legend()

plt.tight_layout()
plt.show()
```

4:11 PM ✓


Sigmoid
Sigmoid
Tanh
Tanh
ReLU
ReLU
Leaky ReLU
Leaky ReLU
Softmax
Softmax
LogSoftmax
LogSoftmax
ELU
ELU
SiLU (Swish)
SiLU (Swish)

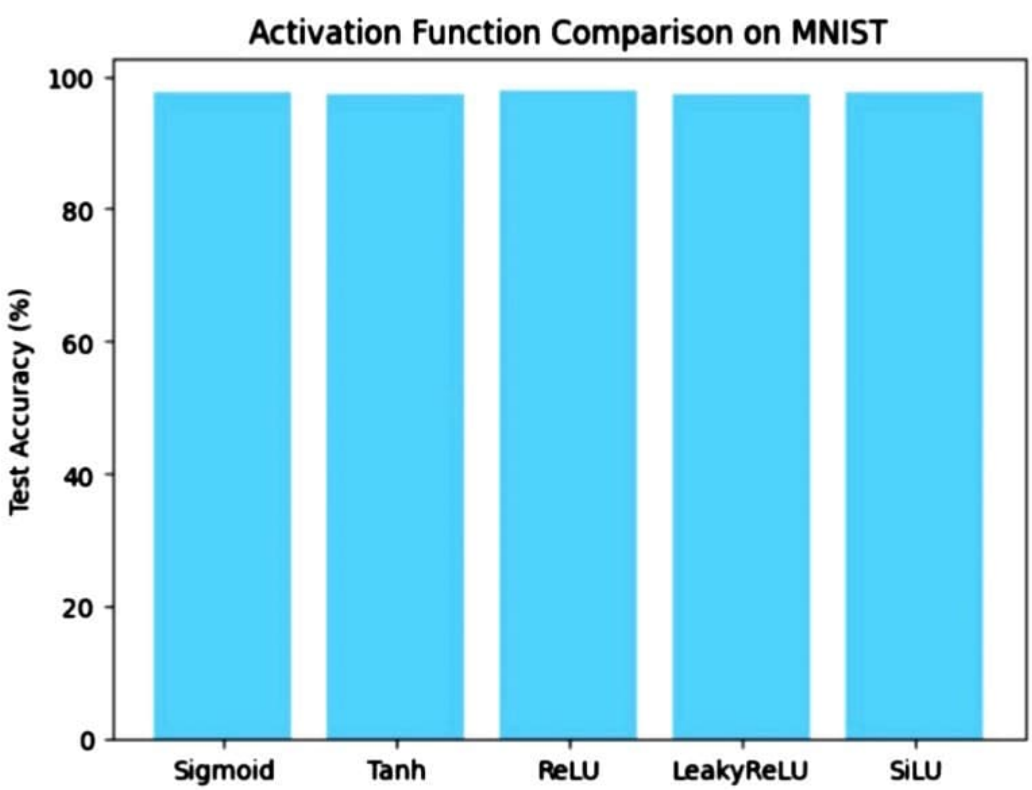
4:11 PM ✓

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Data
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST(root="./data", train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root="./data", train=False, transform=transform, download=True)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)

# Model with configurable activation
class SimpleNN(nn.Module):
    def __init__(self, activation_fn):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.activation = activation_fn

    def forward(self, x):
```

4:11 PM ✓

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 33.4MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.06MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 9.38MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 7.18MB/s]
Training with Sigmoid activation...
Test Accuracy (Sigmoid): 97.74%

Training with Tanh activation...
Test Accuracy (Tanh): 97.24%

Training with ReLU activation...
Test Accuracy (ReLU): 97.85%

Training with LeakyReLU activation...
Test Accuracy (LeakyReLU): 97.45%

Training with SiLU activation...
Test Accuracy (SiLU): 97.62%

=== Final Comparison ===
Sigmoid: 97.74%
Tanh: 97.24%
ReLU: 97.85%
LeakyReLU: 97.45%
SiLU: 97.62%
```

4:12 PM



4:12 PM