[4]
21s

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Step 1: Prepare Dataset (sequence of numbers)
data = np.array([i for i in range(1, 51)])  # numbers 1 to 50

# Function to create input-output pairs
def create_sequences(seq, n_steps):
    X, y = [], []
    for i in range(len(seq) - n_steps):
        X.append(seq[i:i+n_steps])
        y.append(seq[i+n_steps])
    return np.array(X), np.array(y)

n_steps = 5  # length of input sequence
X, y = create_sequences(data, n_steps)

# Reshape for LSTM [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))

# Step 2: Build LSTM Model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# Step 3: Train Model
model.fit(X, y, epochs=200, verbose=0)

# Step 4: Test Prediction
test_input = np.array([46, 47, 48, 49, 50]).reshape((1, n_steps, 1))
predicted = model.predict(test_input, verbose=0)

print("Input sequence: [46, 47, 48, 49, 50]")
print("Predicted next number:", predicted[0][0])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
  super().__init__(**kwargs)
Input sequence: [46, 47, 48, 49, 50]
Predicted next number: 50.89776
```

22/08/25

Exp-5 Study of Activation functions and Their Role in Neural Networks

Aim:

To study different activation functions (sigmoid, Tanh, Relu, Leaky-ReLU and softmax and visualize their effect on an input image using python.

Description:

Activation functions introduce non-linearity in neural network, allowing them to learn complex patterns.

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

- Range (0,1)
- used for probabilities, suffers from vanishing gradients.

Tan h : $f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

- Range : (-1,1)
- Zero centered, better than sigmoid

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

- Range : $[0, \infty]$
- Fast and efficient : risk of dying "ReLU"

Leaky ReLU : $f(x) = x$ if $x>0$ else αx

- Allows small negative values, fixes dying ReLU issue

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- Outputs probabilities (0-1) that sum to 1; used in classification.

Procedure:

1.) Import necessary libraries
2.) Load a sample grayscale image
3.) Implement activation function
4.) Normalize / shift the image for better visualization
5.) Apply each activation function to image.
6.) Display the original and transformed images side by side
7.) Observe how each function modifies pixel intensities

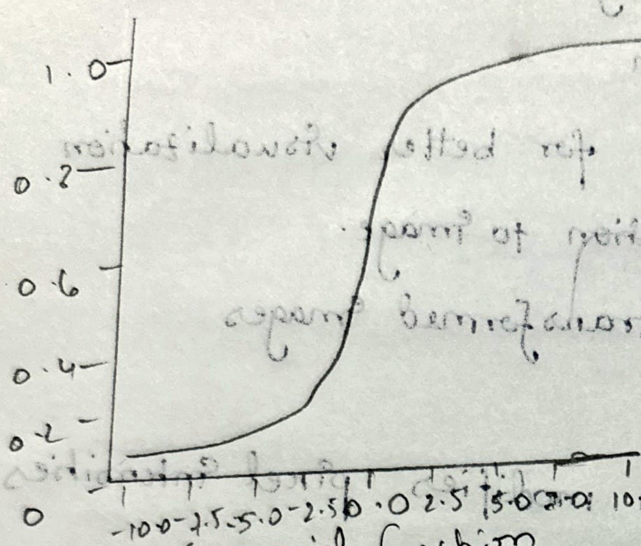
Program:

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def tanh(x):
    return np.tanh(x)

def relu(x):
    return np.maximum (0,x)

def leaky-relu(x, alpha=0.01):
    return np.where (x>0, x, alpha * x)

def softmax(x):
    e_x = np.exp(x-np.max(x))
    return e_x/e_x.sum(axis=0)

x = np.linspace(-10,10,400)

plt.figure(figsize=(12,8))
```

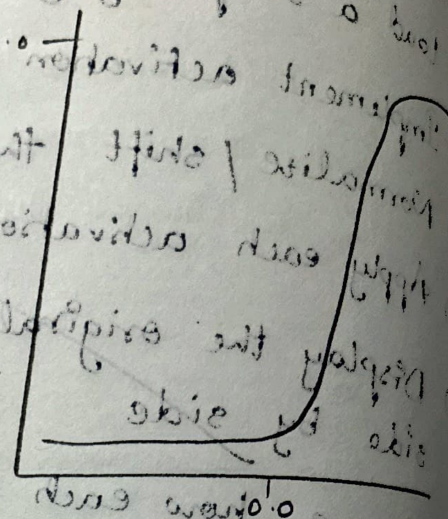
```
plt.subplot(2,2,1)
plt.plot(x, sigmoid(x), 'r')
plt.title("Sigmoid Function")
plt.grid()

plt.subplot(2,2,2)
plt.plot(x, tanh(x), 'g')
plt.title("Tanh Function")
plt.grid()

plt.subplot(2,2,3)
plt.plot(x, relu(x), 'b')
plt.title("ReLU function)
plt.grid()

plt.subplot(2,2,4)
plt.plot(x, relu(x), 'b')
plt.title("ReLU function)
plt.grid()

plt.subplot(2,2,4)
plt.plot(x, leaky_relu(x), 'm')
plt.title("Leaky ReLU function")
plt.grid()

plt.tight_layout()
plt.show()

sample_input = np.array([2.0, 1.0, 0.1])
print("Softmax Output for [2.0, 1.0, 0.1]:", softmax(sample_ip))
```

Result:

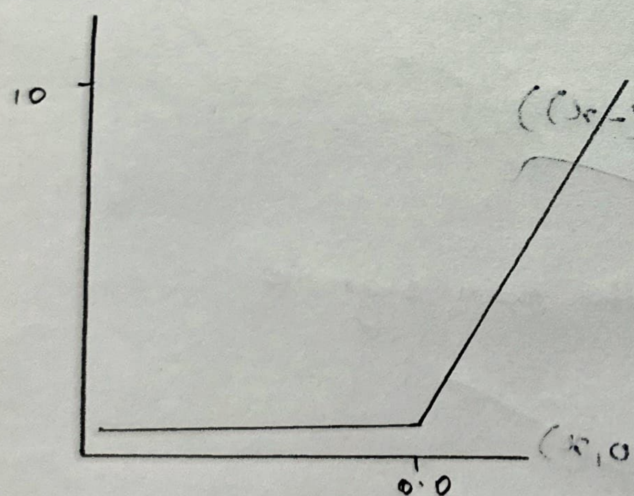
Different Activation functions were implemented, visualized and their role in introducing non-linearity in neural networks was studied.



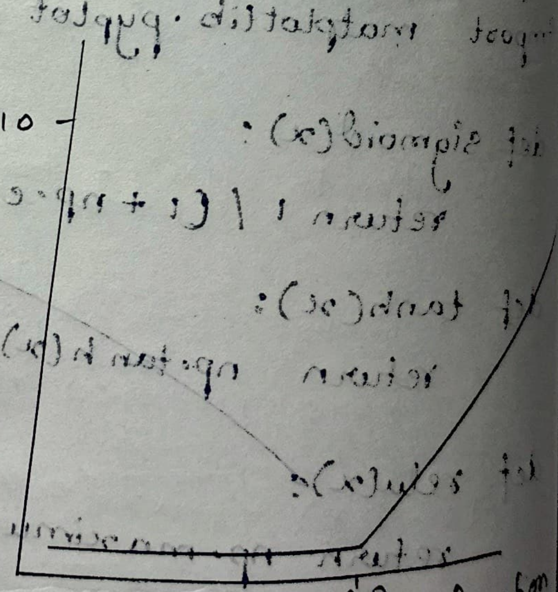
Sigmoid function



Tanh function



Relu function



Leaky Relu function