Put Your Code Results Here:

1. You are required to implement a feedforward neural network with at least 1 hidden layer.

- Pre-activation use Equation 6.7
- Hidden layer (activaiton) us Equation 6.8
- Output layer of hidden layer I use Equation 6.9

```python (below is python command)
a1 = W1 @ x
z1 = activation_function(a1)
z1_aug = np.vstack(([1.0], z1))
y = W2 @ z1_aug
```

2. You must integrate and evaluate five activation functions (Tanh, Hard Tanh, Softplus, ReLU, leakyReLU.).

Tanh

```
Input x (with bias):
 [[1.  0.5 0.2 0.1]]
Hidden pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Hidden activation z1:
 [[0.21651806 0.13909245 0.1194273  0.14888503]]
Hidden layer with bias z1_aug:
 [[1.         0.21651806 0.13909245 0.1194273  0.14888503]]
Final output y:
 [[ 0.32564833 -0.05383076]]
```

Hard Tanh

```
Input x (with bias):
 [[1.  0.5 0.2 0.1]]
Hidden pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Hidden activation z1:
 [[0.22 0.14 0.12 0.15]]
Hidden layer with bias z1_aug:
 [[1.   0.22 0.14 0.12 0.15]]
Final output y:
 [[ 0.327 -0.052]]
```

Softplus

```
Input x (with bias):
 [[1.  0.5 0.2 0.1]]
Hidden pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Hidden activation z1:
 [[0.80918502 0.76559518 0.7549461  0.77095705]]
Hidden layer with bias z1_aug:
 [[1.         0.80918502 0.76559518 0.7549461  0.77095705]]
Final output y:
 [[0.82076474 0.43204936]]
```

ReLU

```
Input x (with bias):
 [[1.  0.5 0.2 0.1]]
Hidden pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Hidden activation z1:
 [[0.22 0.14 0.12 0.15]]
Hidden layer with bias z1_aug:
 [[1.   0.22 0.14 0.12 0.15]]
Final output y:
 [[ 0.327 -0.052]]
```

Leaky ReLU

```
Input x (with bias):
 [[1.  0.5 0.2 0.1]]
Hidden pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Hidden activation z1:
 [[0.22 0.14 0.12 0.15]]
Hidden layer with bias z1_aug:
 [[1.   0.22 0.14 0.12 0.15]]
Final output y:
 [[ 0.327 -0.052]]
```

3. Compare the hidden layer outputs from each activation function. (Attach the screenshoot for each activation function)

- **Tanh**: Outputs values between (-1, 1); saturates for extreme inputs, causing gradient vanishing; preserves negative input values as negative outputs.
- **Hard Tanh**: Similar to Tanh but clips values beyond ±1; easily saturates and limits the range strictly between -1 and 1.
- **Softplus**: Outputs strictly positive values; smooth activation without upper bound saturation; negative inputs become small positive outputs.
- **ReLU**: Converts negative values directly to zero, potentially causing inactive neurons ("dying ReLU"); does not saturate for positive inputs.
- **Leaky ReLU**: Addresses the dying ReLU issue by assigning a small slope to negative values; prevents neurons from fully deactivating.

4. After completing your neural network forward pass in code, choose any one activation function (e.g., tanh, ReLU, etc.), and manually calculate the output of the network.

See below page

$[0.1, 0.1, 0.2, 0.3]$ $[1.0, 0.5, 0.2, 0.1]$

$$a1.1 = 0.1 \times 1.0 + 0.1 \times 0.5 + 0.2 \times 0.2 + 0.3 \times 0.1$$

$$= 0.1 + 0.05 + 0.04 + 0.03$$

$$= 0.22$$

$[0.2, -0.3, 0.4, 0.1]$

$$a1.2 = 0.2 \times 1.0 + (-0.3) \times 0.5 + 0.4 \times 0.2 + 0.1 \times 0.1$$

$$= 0.2 - 0.15 + 0.08 + 0.01$$

$$= 0.14$$

$[0.05, 0.2, -0.2, 0.1]$

$$a1.3 = 0.05 \times 1.0 + 0.2 \times 0.5 + (-0.2) \times 0.2 + 0.1 \times 0.1$$

$$= 0.05 + 0.10 - 0.04 + 0.01$$

$$a1.3 = 0.12$$

$$[0.0, 0.3, -0.1, 0.2]$$

$$a_{1,4} = 0.0 \times 1.0 + 0.3 \times 0.5 + (-0.1) \times 0.2 + 0.2 \times 0.1$$

$$a_{1.4} = 0 + 0.15 - 0.02 + 0.02 = 0.15$$

$$\therefore a_1 = \begin{bmatrix} 0.22 \\ 0.14 \\ 0.12 \\ 0.15 \end{bmatrix}$$

ReLU would not change becaz all +

but add biaz node

$$\begin{bmatrix} 1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.22 \\ 0.14 \\ 0.12 \\ 0.15 \end{bmatrix}$$

W_2

$[0.2, 0.3, -0.1, 0.5, 0.1]$

$[1.0, 0.22, 0.14, 0.12, 0.15]$

$= 0.327$

$[-0.2, 0.4, 0.3, -0.1, 0.2]$

$= -0.052$

final output $\begin{bmatrix} 0.327 \\ -0.052 \end{bmatrix}$ #

ReLU