Machine Learning

LABORATORY: Deep Neural Network

NAME:	STUDENT ID#:

Objectives:

- Understand the architecture and flow of a simple feedforward neural network with one hidden layer.
- Manually implement multiple activation functions (tanh, hard tanh, ReLU, softplus, leaky ReLU) using their mathematical definitions.
- Perform forward propagation from scratch using NumPy.

Part 1. Instruction

- Derive and implement a 1-hidden-layer neural network forward pass.
- Use matrix operations to compute pre- and post-activation values.
- Implement and compare different activation functions.
- Analyze how activation functions impact the output.

Part 2. Arithmetic Instructions.

Step		Procedure
1	Neural network Forward Pass	
	• Equation 6.7:	$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
	• Equation 6.8:	$z_j^{(1)} = h(a_j^{(1)})$
	• Equation 6.9:	$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j^{(1)} + w_{k0}^{(2)}$
2	Activation functions	
	• Equation 6.14:	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
	• Equation 6.15:	$h(a) = \max(-1, \min(1, a))$
	• Equation 6.16:	$h(a) = \ln(1 + \exp(a))$
	• Equation 6.17:	$h(a) = \max(0, a)$
	• Equation 6.18:	$h(a) = \max(0, a) + \alpha \min(0, a$

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

Part 3. Data Transfer Instructions.

Step	Procedure

1	import numpy as np
	
	# ------------------------
	# TODO: 1. Define the activation function
	
	def tanh(x):
	  return None
	def hard_tanh(x):
	  return None
	def softplus(x):
	  return None
	def relu(x):
	  return None
	def leaky_relu(x, alpha=0.1):
	  return None

2	# TODO: 2. Change the Activation Function to Test
	activation_function = tanh # <-- Change this to test others
	# Input Vector x (3 features + bias x0)
	x_raw = np.array([[0.5], [0.2], [0.1]]) # (3, 1)
	x = np.vstack(([1.0], x_raw)) # x0 = 1 added for bias

3	#Define Fixed Weights (No randomness)
	
	W1 = np.array([
	  [0.1, 0.1, 0.2, 0.3],
	  [0.2, -0.3, 0.4, 0.1],
	  [0.05, 0.2, -0.2, 0.1],
	  [0.0, 0.3, -0.1, 0.2]
	]) # Shape: (4 hidden, 4 input incl. bias)
	
	W2 = np.array([
	  [0.2, 0.3, -0.1, 0.5, 0.1],
	  [-0.2, 0.4, 0.3, -0.1, 0.2]
	]) # Shape: (2 output, 5 hidden incl. z0 bias)

4	# TODO: 4. Implement Forward Pass (Equations 6.7–6.12)
	
	# Step 1: Compute pre-activation
	a1 = None # <-- Fill this line
	
	# Step 2: Apply activation function
	z1 = None # <-- Fill this line
	
	# Step 3: Add bias node z0 = 1 to hidden activations
	z1_aug = None # <-- Fill this line
	
	# Step 4: Compute output y
	y = None # <-- Fill this line
	
	print("Input x (with bias):\n", x.T)
	print("Hidden pre-activation a1:\n", a1.T)

```
print("Hidden activation z1:\n", z1.T)
print("Hidden layer with bias z1_aug:\n", z1_aug.T)
print("Final output y:\n", y.T)
```

Grading & Submission Instructions

Assignment (30% max):

1. (7.5%) You are required to implement a feedforward neural network with at least 1 hidden layer.
2. (10%) You must integrate and evaluate five activation functions (Tanh, Hard Tanh, Softplus, ReLU, leakyReLU.).
3. (5%) Compare the hidden layer outputs from each activation function. (Attach the screenshoot for each activation function)
4. (7.5%) 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.

Submission :

1. Report: Answer all conceptual questions. Include screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs3 In Class Assignment**). Name your files correctly:
    a. Report: StudentID_Lab3_InClass.pdf
    b. Code: StudentID_Lab3_InClass.py or StudentID_Lab3_InClass.ipynb
4. Deadline: 4:20 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Example Results (Just for references):

```
=== Activation: tanh (6.14) ===
Input x:
 [[1.  0.5 0.2 0.1]]
Pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Post-activation z1:
 [[0.21651806 0.13909245 0.1194273  0.14888503]]
Output y:
 [[ 0.32564833 -0.05383076]]

=== Activation: hard_tanh (6.15) ===
Input x:
 [[1.  0.5 0.2 0.1]]
Pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Post-activation z1:
 [[0.22 0.14 0.12 0.15]]
Output y:
 [[ 0.327 -0.052]]
```

```
=== Activation: ReLU (6.17) ===
Input x:
 [[1.  0.5 0.2 0.1]]
Pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Post-activation z1:
 [[0.22 0.14 0.12 0.15]]
Output y:
 [[ 0.327 -0.052]]

=== Activation: Leaky ReLU (6.18) ===
Input x:
 [[1.  0.5 0.2 0.1]]
Pre-activation a1:
 [[0.22 0.14 0.12 0.15]]
Post-activation z1:
 [[0.22 0.14 0.12 0.15]]
Output y:
 [[ 0.327 -0.052]]
```

Code Results and Answer:

=== Hidden Layer Outputs (z1) for Each Activation Function ===

	Hidden Neuron 1	Hidden Neuron 2	Hidden Neuron 3	Hidden Neuron 4
tanh	0.2165	0.1391	0.1194	0.1489
hard_tanh	0.2200	0.1400	0.1200	0.1500
softplus	0.8092	0.7656	0.7549	0.7710
relu	0.2200	0.1400	0.1200	0.1500
leaky_relu	0.2200	0.1400	0.1200	0.1500

=== Example Forward Pass: 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]]
```

```
=== Example Forward Pass: 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]]

=== Example Forward Pass: 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]]
```

=== Example Forward Pass: 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]]

=== Example Forward Pass: 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]]

```
=== Example Forward Pass: 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]]
```

```
=== Example Forward Pass: 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]]
```

```
=== Example Forward Pass: 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]]
```

```
=== Example Forward Pass: 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]]
```

```
=== Example Forward Pass: 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]]
```

🔍 **Comparison of Hidden Layer Outputs Across Activation Functions**

The table below shows the **hidden layer outputs** (z1) for each activation function tested:

Activation Function	**Hidden Neuron 1**	**Hidden Neuron 2**	**Hidden Neuron 3**	**Hidden Neuron 4**
Tanh	0.2165	0.1391	0.1194	0.1489
Hard Tanh	0.2200	0.1400	0.1200	0.1500

Activation Function	Hidden Neuron 1	Hidden Neuron 2	Hidden Neuron 3	Hidden Neuron 4
Softplus	0.8092	0.7656	0.7549	0.7710
ReLU	0.2200	0.1400	0.1200	0.1500
Leaky ReLU	0.2200	0.1400	0.1200	0.1500

Observations & Analysis

- **Tanh** produces smooth, symmetric outputs between -1 and 1. In this input scenario, it yields slightly smaller values than Hard Tanh due to its smoother saturation.
- **Hard Tanh**, **ReLU**, and **Leaky ReLU** generate nearly identical outputs here—because all pre-activation values (a1) are positive, they behave similarly and simply pass through the input values (or clamp them).
- **Softplus** is the only activation function that always returns positive outputs and tends to increase the value range. It has the highest outputs among all and may lead to more aggressive activation in some use cases.
- While **ReLU**, **Leaky ReLU**, and **Hard Tanh** appear similar in this specific example, they would behave very differently when handling **negative input values**—which isn't evident here because all a1 values are positive.

Step 1: Hidden pre-activation $a_1 = W_1 \cdot x$

$$x = \begin{bmatrix} 1.0 \\ 0.5 \\ 0.2 \\ 0.1 \end{bmatrix} \quad W_1 = 4 \times 4 \Rightarrow a_1 = \begin{bmatrix} 0.22 \\ 0.14 \\ 0.12 \\ 0.15 \end{bmatrix}$$

Step 2: Apply `tanh`:

$$z_1 = \tanh(a_1) = \begin{bmatrix} \tanh(0.22) \approx 0.2165 \\ \tanh(0.14) \approx 0.1391 \\ \tanh(0.12) \approx 0.1194 \\ \tanh(0.15) \approx 0.1489 \end{bmatrix}$$

Step 3: 加入 bias node $z_0 = 1$

$$z_1^{aug} = \begin{bmatrix} 1.0 \\ 0.2165 \\ 0.1391 \\ 0.1194 \\ 0.1489 \end{bmatrix}$$

Step 4: 輸出層：

$$y = W_2 \cdot z_1^{aug} \Rightarrow \begin{bmatrix} 0.2 & 0.3 & -0.1 & 0.5 & 0.1 \\ -0.2 & 0.4 & 0.3 & -0.1 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 1.0 \\ 0.2165 \\ 0.1391 \\ 0.1194 \\ 0.1489 \end{bmatrix} = \begin{bmatrix} 0.3256 \\ -0.0538 \end{bmatrix}$$