



Machine Learning

LABORATORY: Deep Neural Network

NAME: 張子中

STUDENT ID#: 313605013

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 <ul style="list-style-type: none"> • Equation 6.7: • Equation 6.8: • Equation 6.9: 	$a_j^{(1)} = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$ $z_j^{(1)} = h(a_j^{(1)})$ $a_k^{(2)} = \sum_{j=1}^M w_{kj}^{(2)} z_j^{(1)} + w_{k0}^{(2)}$
2	Activation functions <ul style="list-style-type: none"> • Equation 6.14: • Equation 6.15: • Equation 6.16: • Equation 6.17: • Equation 6.18: 	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$ $h(a) = \max(-1, \min(1, a))$ $h(a) = \ln(1 + \exp(a))$ $h(a) = \max(0, a)$ $h(a) = \max(0, a) + \alpha \min(0, a)$



Part 3. Data Transfer Instructions.

Step	Procedure
1	<pre> 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 </pre>
2	<pre> # 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 </pre>
3	<pre> # 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) </pre>
4	<pre> # 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) </pre>



```
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 screenshot 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:

```

=== Activation Function: 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]]

```

```

=== Activation Function: 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]]

```

```

=== Activation Function: 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]]

```

```

=== Activation Function: 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]]

```

```

=== Activation Function: 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]]

```

```

# Manually calculate the relu
input = [1.  0.5 0.2 0.1]

W1 =
[[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]]

W2 =
[[0.2,  0.3, -0.1, 0.5,  0.1],
 [-0.2, 0.4,  0.3, -0.1, 0.2]]

a1 = W1 @ input

a1 = [0.22 0.14 0.12 0.15] # (4,1)

z1 = relu(a1) = np.maximum(0, a1) # so there will be same

zi = [0.22 0.14 0.12 0.15] # (4,1)

add bias [1.0]

zi_aug = [1.  0.22 0.14 0.12 0.15] # (5,1)

output = W2 @ zi_aug

```

