1. Write the Python code to implement a single neuron.

import numpy as np

**def single\_neuron(input\_data, weights, bias):**

**# Compute the weighted sum and add bias**

**weighted\_sum = np.dot(input\_data, weights) + bias**

**# Apply activation function (e.g., sigmoid)**

**output = 1 / (1 + np.exp(-weighted\_sum))**

**return output**

1. Write the Python code to implement ReLU.

**def relu(x):**

**return max(0, x)**

1. Write the Python code for a dense layer in terms of matrix multiplication.

import numpy as np

**def dense\_layer(input\_data, weights, bias):**

**return np.dot(input\_data, weights) + bias**

1. Write the Python code for a dense layer in plain Python (that is, with list comprehensions

and functionality built into Python).

**def dense\_layer\_plain(input\_data, weights, bias):**

**output = [sum([input\_data[j] \* weights[j][i] for j in range(len(input\_data))]) + bias[i] for i in range(len(bias))]**

**return output**

1. What is the “hidden size” of a layer?

**The hidden size of a layer refers to the number of neurons or units in that layer. It represents the dimensionality of the layer's output.**

1. What does the t method do in PyTorch?

**In PyTorch, the t method is used to transpose a tensor. It switches the dimensions of the tensor, effectively swapping rows with columns.**

1. Why is matrix multiplication written in plain Python very slow?

**Matrix multiplication in plain Python is slow because Python's built-in list operations and iteration are not optimized for large-scale numerical operations. Libraries like NumPy and PyTorch use highly optimized C or CUDA implementations to perform matrix multiplication efficiently.**

1. In matmul, why is ac==br?

**In matrix multiplication (e.g., np.matmul or torch.matmul), the inner dimensions of the two matrices must be equal for the operation to be valid. In the expression matmul(a, b), a has dimensions (a\_rows, a\_cols) and b has dimensions (b\_rows, b\_cols). To perform matrix multiplication, a\_cols must be equal to b\_rows (i.e., a\_cols == b\_rows).**

1. In Jupyter Notebook, how do We measure the time taken for a single cell to execute?

**We can use the %%time magic command at the beginning of a cell in Jupyter Notebook to measure the t%%time**

**# Our code to measure execution timeime taken for the cell to execute. For example:**

1. What is elementwise arithmetic?

**Elementwise arithmetic refers to performing arithmetic operations (addition, subtraction, multiplication, division, etc.) element by element, rather than matrix-wide. It is a fundamental operation in numerical computing.**

1. Write the PyTorch code to test whether every element of a is greater than the

corresponding element of b.

**import torch**

**a = torch.tensor([1, 2, 3])**

**b = torch.tensor([2, 2, 2])**

**result = a > b**

**print(result)**

1. What is a rank-0 tensor? How do We convert it to a plain Python data type?

**A rank-0 tensor in PyTorch is essentially a scalar, a single numeric value.**

**You can convert a rank-0 tensor to a plain Python data type (e.g., float, int) using the .item() method. For example:**

**import torch**

**tensor\_scalar = torch.tensor(5)**

**python\_scalar = tensor\_scalar.item()**

1. How does elementwise arithmetic help us speed up matmul?

**Elementwise arithmetic helps speed up matmul when applied to individual elements or subsets of matrices. This allows parallelization of smaller computations and can make better use of hardware accelerators like GPUs.**

1. What are the broadcasting rules?

**Broadcasting is a set of rules in NumPy and PyTorch that allow you to perform elementwise operations on arrays (tensors) with different shapes. It automatically expands or duplicates dimensions to make the arrays compatible for elementwise operations.**

15. What is expand\_as? Show an example of how it can be used to match the results of

broadcasting.

**The expand\_as method is used to expand the dimensions of a tensor to match the dimensions of another tensor. It is often used in broadcasting to ensure that two tensors can be elementwise operated on without explicit dimension manipulation. Here's an example:**

**import torch**

**a = torch.tensor([1, 2, 3])**

**b = torch.tensor([[10], [20], [30]])**

**a\_expanded = a.expand\_as(b)**

**result = a\_expanded \* b**