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

class Neuron:

def \_\_init\_\_(self, weights, bias):

self.weights = weights

self.bias = bias

def activate(self, inputs):

# Check if the number of weights matches the number of inputs

if len(inputs) != len(self.weights):

raise ValueError("Number of inputs must match number of weights.")

weighted\_sum = sum(w \* x for w, x in zip(self.weights, inputs))

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

return output

neuron = Neuron([0.5, 0.2, 0.1], -0.3)

inputs = [1.0, 2.0, 3.0]

output = neuron.activate(inputs)

print(output)

1. Write the Python code to implement ReLU.

def relu(x):

return max(0, x)

result = relu(5)

print(result) # Output: 5 (since 5 is positive)

result = relu(-2)

print(result) # Output: 0 (since -2 is less than 0)

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

import numpy as np

class DenseLayer:

def \_\_init\_\_(self, weights, biases):

self.weights = weights

self.biases = biases

def forward(self, inputs):

return np.dot(inputs, self.weights) + self.biases

weights = np.array([[0.5, 0.2], [-0.3, 0.1], [0.2, -0.2]])

biases = np.array([0.1, -0.1])

layer = DenseLayer(weights, biases)

inputs = np.array([1.0, 2.0, 3.0])

output = layer.forward(inputs)

print(output)

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(inputs, weights, biases):

if len(inputs) != len(weights[0]):

raise ValueError("Number of inputs must match the number of weights.")

return [sum(w \* x for w, x in zip(weights[i], inputs)) + biases[i] for i in range(len(weights))]

weights = [[0.5, 0.2], [-0.3, 0.1], [0.2, -0.2]]

biases = [0.1, -0.1]

inputs = [1.0, 2.0, 3.0]

output = dense\_layer(inputs, weights, biases)

print(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. For example, in a neural network, you might have an input layer with 10 input features, a hidden layer with 20 neurons, and an output layer with 5 output neurons. In this case, the hidden size of the layer is 20.

1. What does the t method do in PyTorch?

In PyTorch, the t method (short for transpose) is used to compute the transpose of a tensor. It swaps the dimensions of a tensor. For example, if you have a 2D tensor a, you can get its transpose using a.t().

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

Matrix multiplication written in plain Python is slow because it involves nested loops to iterate through the elements of matrices, resulting in O(n^3) time complexity for two n x n matrices. Libraries like NumPy or optimized matrix libraries in other languages (e.g., BLAS) use highly efficient algorithms and low-level optimizations to perform matrix multiplication much faster.

1. In matmul, why is ac==br?

In matmul, ac represents the number of columns in matrix A, and br represents the number of rows in matrix B. To perform matrix multiplication, the number of columns in A must match the number of rows in B, so ac must be equal to br.

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

In Jupyter Notebook, you can measure the time taken for a single cell to execute using the %time or %timeit magic commands. For example, you can use %time before running a cell to measure the time it takes to execute the code in that cell.

1. What is elementwise arithmetic?

Elementwise arithmetic refers to performing arithmetic operations (addition, subtraction, multiplication, division, etc.) element by element on arrays or tensors. It means that corresponding elements in two arrays are operated on together, and the result is an array of the same shape. For example, adding two arrays elementwise results in an array where each element is the sum of the corresponding elements from the two input arrays.

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([0, 2, 4])

result = torch.all(a > b)

print(result)

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

import torch

x = torch.tensor(42)

x\_as\_python = x.item()

print(x\_as\_python)

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

Elementwise arithmetic allows us to perform arithmetic operations on corresponding elements of two arrays or tensors simultaneously. This helps speed up matmul because each element of the resulting matrix can be computed independently based on elementwise multiplication and addition, reducing the need for nested loops.

1. What are the broadcasting rules?

Broadcasting rules in NumPy and other libraries define how arrays of different shapes can be combined in elementwise operations. Broadcasting allows you to perform operations on arrays that don't have exactly the same shape. The smaller array is "broadcasted" over the larger array so that they have compatible shapes for elementwise operations.

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

import torch

a = torch.tensor([1, 2, 3]) # Shape: (3,)

b = torch.tensor([[10], [20], [30]]) # Shape: (3, 1)

expanded\_a = a.unsqueeze(1).expand\_as(b)

print(expanded\_a)