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

import numpy as np

class Neuron:

def \_\_init\_\_(self, input\_size):

self.weights = np.random.randn(input\_size)

self.bias = np.random.randn()

def forward(self, inputs):

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

**2. Write the Python code to implement ReLU.**

def relu(x):

return max(0, x)

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

import numpy as np

class DenseLayer:

def \_\_init\_\_(self, input\_size, output\_size):

self.weights = np.random.randn(input\_size, output\_size)

self.bias = np.random.randn(output\_size)

def forward(self, inputs):

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

**4. Write the Python code for a dense layer in plain Python.**

class DenseLayer:

def \_\_init\_\_(self, input\_size, output\_size):

self.weights = [[random.random() for \_ in range(output\_size)] for \_ in range(input\_size)]

self.bias = [random.random() for \_ in range(output\_size)]

def forward(self, inputs):

output = [sum(inputs[i] \* self.weights[i][j] for i in range(len(inputs))) + self.bias[j] for j in range(len(self.bias))]

return output

**5. What is the "hidden size" of a layer?**

* The **hidden size** refers to the number of neurons or units in a hidden layer of a neural network.

**6. What does the t method do in PyTorch?**

* The t method in PyTorch transposes a 2D tensor (matrix). For example:

import torch

x = torch.tensor([[1, 2], [3, 4]])

x.t() # Returns tensor([[1, 3], [2, 4]])

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

* Plain Python uses interpreted loops, which are not optimized for numerical computations. Libraries like NumPy and PyTorch use highly optimized C and CUDA backends for fast matrix operations.

**8. In matmul, why is ac == br?**

* In matrix multiplication, the number of columns in the first matrix (a) must equal the number of rows in the second matrix (b) for the operation to be valid. This ensures the dimensions align for the dot product.

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

* Use the %%time magic command at the beginning of the cell:

%%time

# code here

**10. What is elementwise arithmetic?**

* Elementwise arithmetic involves applying an operation (e.g., addition, multiplication) to corresponding elements of two tensors of the same shape. For example:

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

b = torch.tensor([4, 5, 6])

a + b # Returns tensor([5, 7, 9])

**11. 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 = a > b # Returns tensor([True, False, False])

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

* A **rank-0 tensor** is a scalar (a single value with no dimensions). To convert it to a plain Python data type:

import torch

x = torch.tensor(42)

x.item() # Returns 42 (Python int)

**13. How does elementwise arithmetic help us speed up matmul?**

* Elementwise arithmetic allows for parallelized operations on tensors, which can be efficiently executed on GPUs. This reduces the computational overhead compared to using explicit loops.

**14. What are the broadcasting rules?**

* Broadcasting rules allow elementwise operations on tensors of different shapes by automatically expanding their dimensions to match:
  1. Align dimensions from the right.
  2. Expand dimensions of size 1 to match the other tensor.
  3. If one tensor has fewer dimensions, add dimensions of size 1 to the left.

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

* expand\_as expands a tensor to match the shape of another tensor. For example:

import torch

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

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

a.expand\_as(b) # Returns tensor([[1, 2, 3], [1, 2, 3], [1, 2, 3]])