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

**import numpy as np**

**class Neuron:**

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

**self.weights = weights**

**self.bias = bias**

**def activate(self, inputs):**

**# Calculate the weighted sum of the inputs and the neuron's weights, plus the bias**

**z = np.dot(inputs, self.weights) + self.bias**

**# Apply the activation function (sigmoid in this case)**

**return 1.0 / (1.0 + np.exp(-z))**

**# Example usage:**

**neuron = Neuron(weights=[0.1, 0.2, 0.3], bias=0.2)**

**output = neuron.activate([1.0, 2.0, 3.0])**

1. Write the Python code to implement ReLU.

**def relu(x):**

**return max(0, x)**

**# Example usage:**

**output = relu(1.5) # output will be 1.5**

**output = relu(-0.5) # output will be 0**

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

**import numpy as np**

**class DenseLayer:**

**def \_\_init\_\_(self, num\_inputs, num\_neurons, activation\_function):**

**# Initialize weights and biases for the layer**

**self.weights = np.random.uniform(size=(num\_inputs, num\_neurons))**

**self.biases = np.random.uniform(size=(1, num\_neurons))**

**# Store the activation function to use for this layer**

**self.activate = activation\_function**

**def forward(self, inputs):**

# Calculate the weighted sum of the inputs and the layer's weights, plus the biases

z = np.dot(inputs, self.weights) + self.biases

# Apply the activation function to the weighted sum

return self.activate(z)

1. Write the Python code for a dense layer in plain Python (that is, with list comprehensions and functionality built into Python).
2. **What is the “hidden size” of a layer?**

The hidden size of a layer refers to the number of units or neurons that are present in that layer. This is also often referred to as the dimensionality of the layer, and it is a key hyperparameter that determines the capacity of the network.

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

In PyTorch, the t method is used to transpose a tensor. This means that it will rearrange the dimensions of the tensor according to a specified permutation.

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

Matrix multiplication in plain Python is slow because Python is an interpreted language, which means that the code is not compiled to machine code before it is executed. This means that each operation in the code has to be interpreted and executed by the Python runtime, which can be slow compared to compiled languages like C++. Additionally, Python is not well optimized for numerical operations like matrix multiplication, so it can be even slower than other interpreted languages when working with large matrices.

1. In matmul, why is ac==br?
2. In Jupyter Notebook, how do you measure the time taken for a single cell to execute?
3. What is elementwise arithmetic?
4. Write the PyTorch code to test whether every element of a is greater than the corresponding element of b.
5. What is a rank-0 tensor? How do you convert it to a plain Python data type?
6. How does elementwise arithmetic help us speed up matmul?
7. What are the broadcasting rules?
8. What is expand\_as? Show an example of how it can be used to match the results of broadcasting.