**1. How does unsqueeze help us to solve certain broadcasting problems?**

* unsqueeze adds a new dimension to a tensor, which can help align tensors for broadcasting. For example, if you have a vector of size (3,) and want to add it to a matrix of size (3, 3), you can use unsqueeze to expand the vector to (3, 1) or (1, 3) so that broadcasting rules apply correctly.

**2. How can we use indexing to do the same operation as unsqueeze?**

* You can use None in indexing to add a new dimension. For example:

import torch

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

x\_unsqueezed = x[:, None] # Adds a new dimension, resulting in shape (3, 1)

**3. How do we show the actual contents of the memory used for a tensor?**

* Use the storage() method to access the underlying memory of a tensor:

import torch

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

print(x.storage())

**4. When adding a vector of size 3 to a matrix of size 3×3, are the elements of the vector added to each row or each column of the matrix?**

* The elements of the vector are added to each **row** of the matrix. For example:

import torch

matrix = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

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

result = matrix + vector # Adds vector to each row of the matrix

print(result)

**5. Do broadcasting and expand\_as result in increased memory use? Why or why not?**

* **No**, broadcasting and expand\_as do not result in increased memory use. They create a view of the tensor without copying data, making the operation memory-efficient.

**6. Implement matmul using Einstein summation.**

* Use torch.einsum to perform matrix multiplication:

import torch

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

b = torch.tensor([[5, 6], [7, 8]])

result = torch.einsum('ik,kj->ij', a, b) # Matrix multiplication

print(result)

**7. What does a repeated index letter represent on the lefthand side of einsum?**

* A repeated index letter on the lefthand side of einsum indicates a **diagonal** or a **trace operation**, where the same index is used for summing along that dimension.

**8. What are the three rules of Einstein summation notation? Why?**

* **Rules**:
  1. **Repeated indices are summed over**: Indices that appear twice in the expression are summed.
  2. **Free indices define the output shape**: Indices that appear only once define the shape of the output.
  3. **Order of indices matters**: The order of indices in the output determines the shape and layout of the result.
* **Why**: These rules allow concise and flexible specification of tensor operations, including matrix multiplication, dot products, and more.

**9. What are the forward pass and backward pass of a neural network?**

* **Forward pass**: Computes the output of the network given an input, propagating data through each layer.
* **Backward pass**: Computes gradients of the loss with respect to the model's parameters using backpropagation, which is used to update the weights.

**10. Why do we need to store some of the activations calculated for intermediate layers in the forward pass?**

* Activations are stored because they are needed to compute gradients during the backward pass. Without storing them, the network would have to recompute these values, which would be inefficient.

**11. What is the downside of having activations with a standard deviation too far away from 1?**

* If activations have a standard deviation too far from 1, it can lead to **vanishing or exploding gradients**, making training unstable or slow.

**12. How can weight initialization help avoid this problem?**

* Proper weight initialization (e.g., Xavier or He initialization) ensures that the activations have a standard deviation close to 1, which helps maintain stable gradients and speeds up training.