1. How does unsqueeze help us to solve certain broadcasting problems?
2. How can we use indexing to do the same operation as unsqueeze?
3. How do we show the actual contents of the memory used for a tensor?
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? (Be sure to check your answer by running this code in a notebook.)
5. Do broadcasting and expand\_as result in increased memory use? Why or why not?
6. Implement matmul using Einstein summation.
7. What does a repeated index letter represent on the lefthand side of einsum?
8. What are the three rules of Einstein summation notation? Why?
9. What are the forward pass and backward pass of a neural network?
10. Why do we need to store some of the activations calculated for intermediate layers in the forward pass?
11. What is the downside of having activations with a standard deviation too far away from 1?
12. How can weight initialization help avoid this problem?

Answer:

1. The **unsqueeze** function in PyTorch allows us to add dimensions of size 1 to a tensor, which can be useful in solving certain broadcasting problems. Broadcasting is a technique used to apply operations between tensors with different sizes, by expanding the smaller tensor to match the size of the larger one. However, in some cases, the tensors may have different numbers of dimensions, which can lead to broadcasting errors. By adding dimensions with size 1 using **unsqueeze**, we can adjust the shapes of the tensors so that they can be broadcasted together.
2. We can use indexing to achieve the same result as **unsqueeze** by using the **None** keyword to add a new dimension of size 1 to a tensor. For example, given a tensor **x** of shape (3,), we can add a new dimension of size 1 to get a tensor of shape (3, 1) using the following code: **x[:, None]** or **x.unsqueeze(1)**.
3. To show the actual contents of the memory used for a tensor in PyTorch, we can use the **numpy** method of the tensor object and call the **array** method. For example, if **x** is a PyTorch tensor, we can view its contents using **x.numpy().array**.
4. When adding a vector of size 3 to a matrix of size 3x3, the elements of the vector are added to each row of the matrix. This is because of broadcasting rules, which match the dimensions of the vector with the corresponding dimensions of the matrix, and repeat the vector along the rows.
5. Both broadcasting and **expand\_as** do not result in increased memory use, because they do not create new tensors, but rather operate on the existing ones. They only modify the metadata of the tensor, such as the size and stride parameters, to change the way the tensor is viewed and accessed.

6.

import torch

# define two matrices

A = torch.randn((3, 4))

B = torch.randn((4, 5))

# perform matrix multiplication using Einstein summation

C = torch.einsum('ij, jk -> ik', A, B)

1. A repeated index letter on the left-hand side of an **einsum** expression represents a summation over that index. For example, in the expression **'ij, jk -> ik'**, the **j** index appears twice, which means that a summation over the **j** axis is performed. This is equivalent to a matrix multiplication operation.
2. The three rules of Einstein summation notation are as follows:
   * Repeated indices in the input tensors indicate a summation over that index.
   * If an index appears on the left-hand side of the expression but not on the right-hand side, it is being contracted and will not appear in the output tensor.
   * If an index appears on the right-hand side but not on the left-hand side, it is being broadcasted and will be replicated along that dimension in the output tensor.

These rules help to specify the input and output shapes of the tensor operations in a concise and expressive way.

1. The forward pass of a neural network involves passing input data through the network's layers to generate predictions or outputs. During the forward pass, each layer applies a linear transformation (matrix multiplication) followed by a non-linear activation function to its input, and passes the result to the next layer. The final output is then compared to the desired output using a loss function, which measures the difference between the predicted and actual values.
2. We need to store some of the activations calculated for intermediate layers in the forward pass of a neural network because they are used in the backward pass (backpropagation) to compute gradients with respect to the network parameters. During backpropagation, the gradients are computed starting from the loss function and propagating backwards through the layers of the network using the chain rule of calculus. The intermediate activations are needed to compute the gradients of the parameters at each layer, which depend on the activations of the previous layer.
3. The downside of having activations with a standard deviation too far away from 1 is that it can lead to issues with the training of the neural network. When the standard deviation of the activations is too high, it can cause the gradients to explode, meaning that they become very large and cause the weights to update in a way that makes the model unstable. On the other hand, when the standard deviation of the activations is too low, it can cause the gradients to vanish, meaning that they become very small and make it difficult for the model to learn.
4. Weight initialization can help avoid the problem of activations with a standard deviation too far away from 1 by setting the initial values of the weights to appropriate values. One common approach is to use Gaussian initialization, which sets the weights to random values drawn from a normal distribution with mean 0 and standard deviation proportional to the square root of the number of inputs to the neuron. This helps to ensure that the activations have a reasonable standard deviation, which can help to prevent issues with exploding or vanishing gradients during training. Other weight initialization methods, such as Xavier initialization or He initialization, also aim to set the initial values of the weights to appropriate values that take into account the structure of the network and the activation functions used.