**Assignment - 12**

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

Ans: unsqueeze adds a dimension to a tensor, which can help align tensor shapes for broadcasting operations. It effectively increases the rank of the tensor, allowing it to match the shape of another tensor for elementwise operations.

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

Ans:

import torch

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

unsqueezed\_tensor = tensor[:, None] # or tensor[:, numpy.newaxis]

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

Ans: We can use the .storage() method in PyTorch to show the actual contents of the memory used for a tensor.

1. 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.)

Ans: When adding a vector of size 3 to a matrix of size 3×3, the elements of the vector are added to each column of the matrix. This is because broadcasting aligns the shapes of the vector and the matrix, allowing for elementwise addition.

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

Ans: No, broadcasting and expand\_as do not result in increased memory use. They are memory-efficient operations that enable elementwise operations between tensors of different shapes without explicitly copying data. Broadcasting allows for computation to be performed on-the-fly, while expand\_as modifies the tensor's shape without duplicating the underlying data.

1. Implement matmul using Einstein summation.

Ans:

import torch

a = torch.randn(3, 4)

b = torch.randn(4, 5)

c = torch.einsum('ij,jk->ik', a, b) # Equivalent to torch.matmul(a, b)

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

Ans: A repeated index letter on the lefthand side of einsum represents a contraction along that axis in the resulting tensor. It indicates that the corresponding dimensions will be summed over during the operation.

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

Ans: The three rules of Einstein summation notation are:

Repeated indices imply summation: When an index appears more than once in an expression, it indicates summation over that index.

An index must appear exactly twice: Each index in the expression must appear exactly twice, once as a subscript and once as a superscript.

Each index has a range: Each index in the expression has a range of values corresponding to the dimensions of the tensors involved.

These rules help succinctly represent complex tensor operations and enable efficient computation by specifying the operations without explicitly iterating over tensor elements.

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

Ans: The forward pass of a neural network involves propagating input data through the network's layers, computing intermediate activations, and producing an output prediction.

The backward pass (also known as backpropagation) involves computing gradients of the loss function with respect to the network's parameters using the chain rule of calculus, starting from the output layer and propagating gradients backward through the network. These gradients are then used to update the network's parameters during training.

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

Ans: We need to store some of the activations calculated for intermediate layers in the forward pass because they are required during the backward pass for computing gradients. These activations serve as inputs to the activation functions and are necessary for calculating the gradient of the loss function with respect to the network's parameters using backpropagation.

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

Ans: The downside of having activations with a standard deviation too far away from 1 is that it can lead to vanishing or exploding gradients during training. This can cause slow convergence or instability in the training process, making it difficult for the model to learn effectively.

1. How can weight initialization help avoid this problem?

Ans: Weight initialization techniques such as Xavier initialization or He initialization can help avoid the problem of vanishing or exploding gradients by setting appropriate initial values for the network's weights. These techniques ensure that the activations and gradients propagate effectively through the network during training, leading to more stable and efficient learning.