General Sir John Kotelawala Defence University

Faculty of Computing

Department of Computational Mathematics

Intake 38 – Semester VII

CS4193 – Deep Learning

Individual Assignment -2

Registration Number: D/DBA/21/0018

Name: KHR Pabasara

Submission Date: 08/03/2024

Go through pytorch documentation and explain 25 pytorch tensor functions. Include simple code snippets.

PyTorch

PyTorch is an open-source deep learning framework based on the Python programming language. It provides a dynamic computational graph, allowing users to build and modify their models. PyTorch includes an autograd system for automatic differentiation and supports GPU acceleration. Additionally, it has a comprehensive ecosystem with additional libraries such as TorchVision, TorchText, PyTorch Lightning, etc.

Tensors

Tensor in PyTorch is a fundamental data structure, like a NumPy array. PyTorch provides a rich set of tensor functions and operations for performing various mathematical and computational tasks. They can have any number of dimensions and hold various data types.

PyTorch offers variety of functions to perform operations on tensors. They enable efficient calculations and manipulations required for deep learning tasks.

PyTorch tensor functions

1. Torch.acos()
   When we use this function, we can compute the inverse cosine of each element in the input tensor. This function can return new tensor with the arccosine of each element.

```
import torch
x = torch.tensor([0.5, -0.8, 1.0])

# Apply acos
result = torch.acos(x)
print(result)

```

```
tensor([1.0472, 2.4981, 0.0000])
```

2. Torch.any()
   This function checks if any element in the input tensor is non-zero along a specific dimension. It gives the output as a boolean value indicating whether any element is true.

```
import torch
x = torch.tensor([0, 0, 1, 0])
# Check if any element is nonzero
result = torch.any(x)
print(result)

```

```
tensor(True)
```

3. torch.bincount()
   Using this function, we can count the occurrences of each value in an input tensor. Usually used for count occurrences of each integer in a tensor

```
import torch

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

```

```
tensor([0, 2, 2, 1, 1])
```

   Output explanation:

   Value 0 does not appear in the input tensor. Therefore, the count is 0.
   Value 1 appears twice in the input tensor. Therefore, the count is 2.
   Value 2 appears twice in the input tensor. Therefore, the count is 2.

4. torch.chunk()
   chunk function splits the input tensor into a specific number of chunks. It returns a tiple of chunks.

```
import torch

# Example tensor
x = torch.tensor([1, 2, 3, 4, 5, 6])

# Split the tensor into chunks
result = torch.chunk(x, chunks=2)
print(result)

```

```
(tensor([1, 2, 3]), tensor([4, 5, 6]))
```

5. torch.clamp()

We use this function to clip the values of the input tensor to be within a specified range. In here values below the minimum are set to the minimum, and values above the maximum are set to the maximum.

```
import torch

# Example tensor
x = torch.tensor([-1, 2, 5, -3, 0])

# Clamp values between 0 and 3
result = torch.clamp(x, min=0, max=3)
print(result)

```

```
tensor([0, 2, 3, 0, 0])
```

6. torch.cross()

This can be used to computes the cross product of two tensors. Specially it is useful for calculating the cross product of 3D vectors.

```
import torch

# Example tensors
a = torch.tensor([1, 2, 3])
b = torch.tensor([4, 5, 6])

# Compute cross product
result = torch.cross(a, b)
print(result)

```

```
tensor([-3,  6, -3])
```

7. torch.diag()
We can extract the diagonal elements of a tensor by utilizing this function. It returns a new 1-dimensional tensor with the diagonal values as the output.

```
import torch

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

# Extract diagonal
result = torch.diag(x)

print(result)

```

```
tensor([1, 5, 9])
```

8. torch.diagonal()
This function is also used to return the diagonal of a tensor or constructs a diagonal tensor. However, this is more flexible than the above function.

```
import torch

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

# Get diagonal
result = torch.diagonal(x)

print(result)

```

```
tensor([1, 5, 9])
```

9. torch.diff()

In this function, it computes the difference between consecutive elements of a tensor along a specified axis. This is useful for calculating discrete differences.

```
import torch

# Example tensor
x = torch.tensor([1, 4, 7, 2, 9])

# Compute differences
result = torch.diff(x)

print(result)

```

```
tensor([ 3,  3, -5,  7])
```

10. torch.divide()

Divide function, performs element-wise division of two tensors or scalar.

```
import torch

# Example tensors
a = torch.tensor([10, 20, 30])
b = torch.tensor([2, 5, 3])

# Perform division
result = torch.divide(a, b)

print(result)

```

```
tensor([ 5.,  4., 10.])
```

11. torch.fix().
    This function can be used to rounds each element of the input tensors to the nearest integer towards zero. It truncates the decimal part.

```python
import torch

# Example tensor
x = torch.tensor([-2.7, 3.5, -1.2, 4.8])

# Apply fix
result = torch.fix(x)
print(result)

```

```
tensor([-2.,  3., -1.,  4.])
```

12. torch.flip()
    We can use this to reverses the order of elements in a tensor along a specified dimension.

```python
import torch

# Example tensor
x = torch.tensor([[1, 2, 3], [4, 5, 6]])

# Flip along the second dimension
result = torch.flip(x, dims=[1])

# Print the result
print(result)

```

```
tensor([[3, 2, 1],
        [6, 5, 4]])
```

13. torch.gather()

This function use to gathers values along a given axis from an input tensor based on index values provided in another tensor.

```
import torch

# Example tensor
x = torch.tensor([[1, 2], [3, 4], [5, 6]])

# Indices to gather
indices = torch.tensor([[0, 1], [1, 0], [0, 1]])

# Gather values
result = torch.gather(x, dim=1, index=indices)

print(result)

```

```
tensor([[1, 2],
        [4, 3],
        [5, 6]])
```

14. torch.greater()

This computes element-wise greater than comparison between two tensors. It returns a tensor of the same shape with boolean values.

```
import torch

# Example tensors
a = torch.tensor([1, 3, 5])
b = torch.tensor([2, 2, 5])

# Perform greater than comparison
result = torch.greater(a, b)

# Print the result
print(result)

```

```
tensor([False,  True, False])
```

15. torch.inverse()

We can use inverse function to computes the inverse of a square matrix. Usually this function can utilize in linear algebra for solving systems of linear equations.

```
import torch

# Example tensor
x = torch.tensor([[4, 7], [2, 6]], dtype=torch.float)

# Compute inverse
result = torch.inverse(x)

print(result)

```

```
tensor([[ 0.6000, -0.7000],
        [-0.2000,  0.4000]])
```

16. torch.log10()

This function computes the base 10 logarithm of the input tensor elementwise. The resulting tensor will have the same shape as the input, with each element representing the base-10 logarithm of the corresponding element in the original tensor.

```
import torch

# Example tensor
x = torch.tensor([1, 10, 100])

# Compute base 10 logarithm
result = torch.log10(x)
print(result)

```

```
tensor([0., 1., 2.])
```

17. torch.lt()

In here, this function performs elementwise less than comparison between two tensors.
The output is a binary tensor, where each element is True if the corresponding elements in the input tensors satisfy the less-than condition and otherwise it is False.

```
import torch

# Example tensors
a = torch.tensor([1, 3, 5])
b = torch.tensor([2, 2, 5])

# Perform less than comparison
result = torch.lt(a, b)
print(result)

```

```
tensor([ True, False, False])
```

18. torch.max()

This function returns the maximum value of all elements in the input tensor. It provides a single scalar output representing the maximum value.

```
import torch

# Example tensor
x = torch.tensor([[1, 2, 3], [4, 5, 6]])

# Compute maximum value
result = torch.max(x)

print(result)

```

```
tensor(6)
```

19. torch.mean()
In here, this function can be used to computes the mean of all elements in the input tensor.

```
import torch

# Example tensor
x = torch.tensor([1.0, 2.0, 3.0, 4.0])

# Compute mean
result = torch.mean(x)

print(result)

```

```
tensor(2.5000)
```

20. torch.narrow()
This function generates a new tensor that is a view of the original tensor, selecting a specific subset of elements along a specified dimension.
This can be useful, if we need to extract a range of elements along a particular dimension of the tensor.

```
import torch

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

# Narrow along dimension 1
result = torch.narrow(x, dim=1, start=1, length=2)

print(result)

```

```
tensor([[2, 3],
        [5, 6],
        [8, 9]])
```

21. torch.reciprocal()
    This computes the element-wise reciprocal of the input tensor. We can use this function to obtain the reciprocal of each element in the tensor, resulting in a new tensor with values representing the multiplicative inverses.

```
import torch

# Example tensor
x = torch.tensor([2.0, 4.0, 8.0])

# Compute reciprocal
result = torch.reciprocal(x)

print(result)

```

```
tensor([0.5000, 0.2500, 0.1250])
```

22. torch.roll()
    This function rolls the elements of a tensor along a given dimension. It means that this can be used to circularly shift the elements of a tensor, along a specified dimension.

```
import torch

# Example tensor
x = torch.tensor([1, 2, 3, 4, 5])

# Roll elements to the right by 2 positions
result = torch.roll(x, shifts=2, dims=0)

print(result)

```

```
tensor([4, 5, 1, 2, 3])
```

23. torch.select()

We can use this to return a new tensor with elements from the input tensor along a specified dimension.

It is helpful to extract targeted data.

```
import torch

# Example tensor
x = torch.tensor([[1, 2], [3, 4], [5, 6]])

# Select elements along dimension 1
result = torch.select(x, dim=1, index=0)

print(result)

```

```
tensor([1, 3, 5])
```

24. torch.unbind()

This function is used to remove a tensor dimension along a specified axis. This function is practical for splitting a tensor along a particular dimension into individual slices. It provides tuple of tensors.

```
import torch

# Example tensor
x = torch.tensor([[1, 2, 3], [4, 5, 6]])

# Unbind along dimension 0
result = torch.unbind(x, dim=0)

print(result)

```

```
(tensor([1, 2, 3]), tensor([4, 5, 6]))
```

25. torch.var()

We use this function to computes the variance of all elements in the input tensor. Variance is a measure of the spread of data, and this function proves useful when there is a need to quantify the extent of variability within a tensor, resulting in a single scalar representing the variance.

```
1 import torch
2
3 # Example tensor
4 x = torch.tensor([1.0, 2.0, 3.0, 4.0])
5
6 # Compute variance
7 result = torch.var(x)
8
9 print(result)
10
```

```
tensor(1.6667)
```

References

[1] “torch. Tensor — PyTorch 2.2 documentation,” *pytorch.org*. https://pytorch.org/docs/stable/tensors.html (accessed Mar. 08, 2024).

[2] “What is PyTorch?” *NVIDIA Data Science Glossary*. https://www.nvidia.com/en-us/glossary/pytorch/

[3] N. Aslam, “Data Representation in Neural Networks- Tensor,” *Analytics Vidhya*, Jul. 25, 2022. https://www.analyticsvidhya.com/blog/2022/07/data-representation-in-neural-networks-tensor/#:~:text=A%20tensor%20can%20be%20a

[4] “PyTorch Tensors | A Complete Guide to PyTorch Tensors,” *EDUCBA*, Mar. 21, 2022. https://www.educba.com/pytorch-tensors/ (accessed Mar. 08, 2024).