Week 1 Questions (220962018)

January 4, 2025

0.1 Week 1 Lab Exercise

[1]:
```
import torch
device = "cuda" if torch.cuda.is_available() else "cpu"
torch.cuda.device_count() , device
```

[1]:
```
(1, 'cuda')
```

0.2 Lab Exercise

0.2.1 1) Illustrate the functions for Reshaping, viewing, stacking, squeezing and unsqueezing of tensors

[2]:
```
x = torch.arange(12)
print("Original Tensor:")
print(x)

# Reshape the tensor
reshaped_x = x.reshape(3, 4)
print("\nReshaped Tensor:")
print(reshaped_x)
```

```
Original Tensor:
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

Reshaped Tensor:
tensor([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]])
```

[3]:
```
# View the tensor in a different shape (same as reshape here)
viewed_x = x.view(3, 4)
print("\nViewed Tensor:")
print(viewed_x)
```

```
Viewed Tensor:
tensor([[ 0,  1,  2,  3],
```

```
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]])
```

[4]:
```
# Stack two tensors along a new dimension (dimension 0 here)
y = torch.ones(12)
stacked = torch.stack((x, y), dim=0)
print("\nStacked Tensor:")
print(stacked)
```

```
Stacked Tensor:
tensor([[ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.]])
```

[5]:
```
# Create a tensor with extra dimensions (e.g., 1x3x1)
z = torch.ones(1, 3, 1)
print("\nTensor with extra dimensions:")
print(z)

# Squeeze to remove dimensions of size 1
squeezed_z = z.squeeze()
print("\nSqueezed Tensor:")
print(squeezed_z)
```

```
Tensor with extra dimensions:
tensor([[[1.],
         [1.],
         [1.]]])

Squeezed Tensor:
tensor([1., 1., 1.])
```

[6]:
```
# Unsqueeze the tensor to add a new dimension at position 0
unsqueezed_z = z.unsqueeze(0)
print("\nUnsqueezed Tensor:")
print(unsqueezed_z)
```

```
Unsqueezed Tensor:
tensor([[[[1.],
          [1.],
          [1.]]]])
```

0.2.2 2) Illustrate the use of torch.permute()

[7]:
```
# Create a tensor with shape (3, 1)
tensor = torch.randn(3, 1)
print("Original tensor:")
print(tensor)

# Permute the tensor to change the order of dimensions to (1, 3)
permuted_tensor = tensor.permute(1, 0)
print("\nPermuted tensor (dimensions reordered to (1, 3)):")
print(permuted_tensor)
```

```
Original tensor:
tensor([[-0.3001],
        [-1.6191],
        [-1.7779]])

Permuted tensor (dimensions reordered to (1, 3)):
tensor([[-0.3001, -1.6191, -1.7779]])
```

0.2.3 3) Illustrate indexing in tensors

Basic Indexing

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

print("Original Tensor:")
print(tensor)

element = tensor[0, 1]
print("\nElement at position (0, 1):", element)

element = tensor[2, 2]
print("\nElement at position (2, 2):", element)
```

```
Original Tensor:
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])

Element at position (0, 1): tensor(2)

Element at position (2, 2): tensor(9)
```

Slicing

[9]:
```
# Slice the first two rows and all columns
sub_tensor = tensor[:2, :]
print("\nFirst two rows:")
print(sub_tensor)
```

```
# Slice all rows and the first two columns
sub_tensor = tensor[:, :2]
print("\nFirst two columns:")
print(sub_tensor)

# Slice specific rows and specific columns (rows 1 and 2, columns 0 and 2)
sub_tensor = tensor[1:3, [0, 2]]
print("\nRows 1-2, Columns 0 and 2:")
print(sub_tensor)
```

```
First two rows:
tensor([[1, 2, 3],
        [4, 5, 6]])

First two columns:
tensor([[1, 2],
        [4, 5],
        [7, 8]])

Rows 1-2, Columns 0 and 2:
tensor([[4, 6],
        [7, 9]])
```

Boolean Mask Indexing

[10]:
```
# Create a mask to select elements greater than 5
mask = tensor > 5
print("\nMask (values greater than 5):")
print(mask)

# Use the mask to index the tensor
filtered_tensor = tensor[mask]
print("\nElements greater than 5:")
print(filtered_tensor)
```

```
Mask (values greater than 5):
tensor([[False, False, False],
        [False, False,  True],
        [ True,  True,  True]])

Elements greater than 5:
tensor([6, 7, 8, 9])
```

Integer Indexing with a List of Indices

[11]:
```
# Select elements at specific indices (0th and 2nd row, 1st and 2nd column)
indexed_tensor = tensor[[0, 2], [1, 2]]
print("\nElements at specific indices (rows 0 and 2, columns 1 and 2):")
print(indexed_tensor)
```

```
Elements at specific indices (rows 0 and 2, columns 1 and 2):
tensor([2, 9])
```

Advanced Indexing (using ellipsis and multiple dimensions)

[12]:
```
# Create a 3D tensor (2x3x3)
tensor_3d = torch.tensor([[[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                          [[10, 11, 12], [13, 14, 15], [16, 17, 18]]])

# Use ellipsis to select all rows and columns for a specific slice
sub_tensor_3d = tensor_3d[0, ..., 1]  # First matrix, all rows, column 1
print("\nFirst matrix, column 1:")
print(sub_tensor_3d)
```

```
First matrix, column 1:
tensor([2, 5, 8])
```

0.2.4 4) Show how numpy arrays are converted to tensors and back again to numpy arrays

[13]:
```
import torch
import numpy as np

# 1. Convert NumPy array to PyTorch tensor
# Create a NumPy array
numpy_array = np.array([[1, 2, 3], [4, 5, 6]])

# Convert it to a PyTorch tensor
tensor = torch.from_numpy(numpy_array)
print("NumPy array converted to PyTorch tensor:")
print(tensor)

# 2. Convert PyTorch tensor back to NumPy array
# Convert the tensor back to a NumPy array
numpy_array_back = tensor.numpy()
print("\nPyTorch tensor converted back to NumPy array:")
print(numpy_array_back)
```

```
NumPy array converted to PyTorch tensor:
tensor([[1, 2, 3],
        [4, 5, 6]])
```

```
PyTorch tensor converted back to NumPy array:
[[1 2 3]
 [4 5 6]]
```

0.2.5 5) Create a random tensor with shape (7,7)

[14]:
```
# Create a random tensor with shape (7, 7)
random_tensor = torch.randn(7, 7)

print(random_tensor)
```

```
tensor([[ 1.2068, -0.5378, -0.8532, -1.3890, -0.6293,  0.4823, -0.0514],
        [-0.8510, -1.4764, -0.9283, -0.3983,  0.3857,  1.1001,  0.4326],
        [-1.9952, -0.2872,  1.0153,  1.2807,  0.9160,  1.5807,  1.6830],
        [-1.2072,  1.1918,  0.9738, -1.0732, -0.0062, -0.3990, -1.1947],
        [ 1.5098, -0.8165, -1.3619,  0.3337,  0.9791, -0.9343,  1.4641],
        [-2.3407, -1.6107, -1.2218, -1.1764, -0.5338,  2.2783,  0.3007],
        [ 0.7723, -0.4188,  0.5436, -1.1967, -0.0619,  0.4634,  0.2983]])
```

0.2.6 6) Perform a matrix multiplication on the tensor from 5 with another random tensor with shape (1,7)

[15]:
```
# Create a random tensor with shape (7, 7)
tensor_7x7 = torch.randn(7, 7)

# Create another random tensor with shape (1, 7)
tensor_1x7 = torch.randn(1, 7)

# Perform matrix multiplication (7x7) * (7x1)
result = torch.matmul(tensor_7x7, tensor_1x7.T)  # Transpose tensor_1x7 to (7,␣
↪1)

print("Result of matrix multiplication:")
print(result)
```

```
Result of matrix multiplication:
tensor([[-2.8388],
        [-0.4565],
        [ 7.4452],
        [ 2.6844],
        [-0.3894],
        [-0.1411],
        [-4.5858]])
```

0.2.7 7) Create two random tensors of shape (2,3) and send them both to the GPU

[17]:
```
# Check if CUDA (GPU support) is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create two random tensors of shape (2, 3)
tensor1 = torch.randn(2, 3)
tensor2 = torch.randn(2, 3)

# Move the tensors to the GPU
tensor1 = tensor1.to(device)
tensor2 = tensor2.to(device)

# Print the tensors and their device location
print("Tensor 1 on device:", tensor1.device)
print(tensor1)
print("\nTensor 2 on device:", tensor2.device)
print(tensor2)
```

```
Tensor 1 on device: cuda:0
tensor([[-0.1455,  1.5434,  0.8633],
        [-0.4537, -0.7770,  2.7843]], device='cuda:0')

Tensor 2 on device: cuda:0
tensor([[-1.0497, -0.4680, -0.1795],
        [-0.8937,  0.3618,  1.5267]], device='cuda:0')
```

0.2.8 8) Perform a matrix multiplication on the tensors you created in 7

[19]:
```
# Perform matrix multiplication (2x3) * (3x2)
result = torch.matmul(tensor1, tensor2.T)

print("Result of matrix multiplication:")
print(result)
```

```
Result of matrix multiplication:
tensor([[-0.7246,  2.0064],
        [ 0.3402,  4.3752]], device='cuda:0')
```

0.2.9 9) Find the maximum and minimum values of the ouput of 8

[20]:
```
# Find the maximum and minimum values in result
max_value = torch.max(result)
min_value = torch.min(result)

print(f"Max value in Result: {max_value}")
print(f"Max value in Result: {min_value}")
```

```
Max value in Result: 4.375241756439209
Max value in Result: -0.7246068716049194
```

0.2.10 10) Find the maximum and minimum index values of the output of 8

[21]:
```python
# Find the maximum and minimum index values in result
max_value = torch.argmax(result)
min_value = torch.argmin(result)

print(f"Max value in Result: {max_value}")
print(f"Max value in Result: {min_value}")
```

```
Max value in Result: 3
Max value in Result: 0
```

0.2.11 11) Make a random tensor with shape(1,1,1,10) and then create a new tensor with all the 1 dimensions removed to be left with a tensor of shape (10). Set the seed to 7 when you create it and print out the first tensor and it's shape as well as the second tensor and it's shape.

[23]:
```python
# Set the seed for reproducibility
torch.manual_seed(7)

# Create a random tensor with shape (1, 1, 1, 10)
tensor1 = torch.randn(1, 1, 1, 10)

# Remove dimensions of size 1 to get shape (10)
tensor2 = tensor1.squeeze()

# Print the first tensor and its shape
print("First tensor (shape (1, 1, 1, 10)):")
print(tensor1)
print(f"Shape of the first tensor: {tensor1.shape}")

# Print the second tensor and its shape
print("\nSecond tensor after squeezing (shape (10)):")
print(tensor2)
print(f"Shape of the second tensor: {tensor2.shape}")
```

```
First tensor (shape (1, 1, 1, 10)):
tensor([[[[-0.1468,  0.7861,  0.9468, -1.1143,  1.6908, -0.8948, -0.3556,
            1.2324,  0.1382, -1.6822]]]])
Shape of the first tensor: torch.Size([1, 1, 1, 10])

Second tensor after squeezing (shape (10)):
tensor([-0.1468,  0.7861,  0.9468, -1.1143,  1.6908, -0.8948, -0.3556,  1.2324,
         0.1382, -1.6822])
Shape of the second tensor: torch.Size([10])
```