PyTorch Tensor Notebook

This notebook will allow you to get practice in utilizing tensors in PyTorch and explore their proerties and uses. Code excersises denoted by a problem number (i.e. Problem #1) will include a task and a code block that asks for your solution. These blocks will be denoted by comments of the form '# YOUR CODE HERE #'. The code immediately following include assertions that are used to check completeness of the response. They will raise an exception if the previous solution is not complete or not correct.

Declaring, Initializing, and Operating on PyTorch Tensors

Reference: The Linux Foundation, "Tensors-PyTorch Tutorials 2.6.0 +cu124 documentation," pytorch.org https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html (accessed Mar. 12, 2025).

In [1]:
```
import torch
import numpy as np
```

Problem #1: Given the data in python list (data) and numpy array format (numpy_data). Initialize a PyTorch tensor with the data named "pt_tensor", using data, and "pt_tensor_numpy", using numpy_data.

In [2]:
```
data = [[3, 6, 9, 12],[7, 14, 21, 28],[9, 18, 27, 36]]
numpy_data = np.array(data)

### BEGIN SOLUTION
pt_tensor = torch.tensor(data)
pt_tensor_numpy = torch.tensor(numpy_data)
### END SOLUTION

assert pt_tensor.shape == (3,4)
assert pt_tensor_numpy.shape == (3,4)
assert (pt_tensor - pt_tensor_numpy).sum().item() == 0
```

Problem #2: Change the data type of the "pt_tensor" PyTorch tensor to a float (hint: use torch.float) and put resulting tensor in "pt_tensor_float".

In [3]:
```
### BEGIN SOLUTION
pt_tensor_float = torch.tensor(pt_tensor.numpy(), dtype = torch.float)
#OR#
pt_tensor_float = pt_tensor.to(dtype=torch.float)
### END SOLUTION

assert pt_tensor_float.dtype == torch.float
```

Problem #3: Create a tensor of dimension size (choose suitable values saved to a variable named "size") initialized with ones, zeros, and random numbers. These tensors should be named "tensor_ones", "tenor_zeros", and "tensor_rand".

In [4]:
```
### BEGIN SOLUTION
size = (3,5,7)
tensor_ones = torch.ones(size)
tensor_zeros = torch.zeros(size)
tensor_rand = torch.rand(size)
### END SOLUTION

assert (tensor_ones == 1).all()
assert (tensor_zeros == 0).all()
assert ((0 <= tensor_rand).logical_and(tensor_rand < 1)).all()
```

Problem #4: Create four random tensors of dimension (m, n, p), (m,n), (n), and (m). Save the dimensions in variables m, n, and p. The tensors should be named "A", "B", "a", and "b", respectively.

In [5]:
```
### BEGIN SOLUTION ###
m, n, p = 3, 5, 9
A = torch.rand(m,n,p) # Hint dimension (m,n,p)
B = torch.rand(m,n) # Hint dimension (m,n)
a = torch.rand(n) # Hint dimension (n)
b = torch.rand(m) # Hint dimension (m)
### END SOLUTION ###

assert (A.shape == (m,n,p)) and ((0 <= A).logical_and(A < 1)).all()
assert (B.shape == (m,n)) and ((0 <= B).logical_and(B < 1)).all()
assert (a.shape[0] == (n)) and ((0 <= a).logical_and(a < 1)).all()
assert (b.shape[0] == (m)) and ((0 <= b).logical_and(b < 1)).all()
```

Problem #5: Using the tensors A, B, a, and b from above. Perform the following matrix calculations $\mathbf{y} = (\mathbf{B}^T\mathbf{b} + \mathbf{a})$ and $\mathbf{Z} = \mathbf{yA}$.

In [6]:
```
### BEGIN SOLUTION
y = (B.T @ b) + a
Z = y @ A
### END SOLUTION

assert y.shape[0] == (n)
assert Z.shape == (m,p)
```

Problem #6: Using the tensors A, B, a, and b from above. Reshape B to dimensions (m,n,1) and perform the following matrix calculation $\mathbf{k} = \mathbf{A} + \mathbf{B}$.

In [7]:
```
### BEGIN SOLUTION
B = B.reshape((m,n,1))
k = A + B
### END SOLUTION

assert B.shape == (m,n,1)
assert k.shape == (m,n,p)
```

Using GPUs with Tensors

Problem #7: Check to see if a cuda device is available, if so set "device" to that torch.device(...).

In [8]:
```
device = torch.device('cpu')

### BEGIN SOLUTION
if torch.cuda.is_available():
    device = torch.device(torch.cuda.current_device())
### END SOLUTION
print(f"Using device - {device}")

assert isinstance(device, torch.device)
assert torch.cuda.is_available() and device == torch.device('cuda') or torch.device
```

```
Using device - cuda:0
```

Problem #8: Create a random tensor, "A" with dimensions (m,n) and "B" with dimensions (n,m) on the current device, GPU if available. Calculate the matrix multiplicaiton **C** = **AB**.

In [9]:
```
### BEGIN SOLUTION
A = torch.rand(m,n, device=device)
B = torch.rand(n,m)
B = B.to(device)
C = A @ B
### END SOLUTION

assert (A.shape == (m,n)) and ((0 <= A).logical_and(A < 1)).all()
assert (B.shape == (n,m)) and ((0 <= B).logical_and(B < 1)).all()
assert A.device == device and B.device == device and C.device == device
assert C.shape == (m,m)
```

Tensor Parameters and Automatic Differentiation

Reference: The Linux Foundation, "Automatic Differentiation wtih torch.autograd - PyTorch Tutorials 2.6.0 +cu124 documentation," pytorch.org https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html (accessed Mar. 13, 2025).

Problem #9: Create a random tensor, "D" with dimensions (p,m) and set as a parameter. Hint: Use requires_grad and the same device as the predefined tensors. Use the defined tensors, "x" and "r" to perform the operation **y** = **D(xB)** + **r** Calculate the mean squared error loss from the given tensor, "y_0", name the result "mse_loss". Average the elements of the resulting vector together. Hint: Use .mean().

In [10]:
```
y_0 = torch.rand(p,device=device)
x = torch.ones(n,device=device)
```

```python
r = torch.rand(p,device=device)

### BEGIN SOLUTION ###
D = torch.rand(p,m,device=device,requires_grad=True)
y = (D @ (x @ B)) + r
mse_loss = ((y-y_0)**2).mean()
### END SOLUTION ###

mse_loss.backward()
print(f"The mse_loss is {mse_loss.item()}.")
print(f"The gradient for parameter 'D' is: {D.grad}")

assert D.shape == (p,m) and ((0 <= D).logical_and(D < 1)).all()
assert y.shape[0] == p
```

```
The mse_loss is 9.678096771240234.
The gradient for parameter 'D' is: tensor([[1.6338, 0.6444, 0.6970],
        [3.5511, 1.4006, 1.5149],
        [3.1871, 1.2570, 1.3596],
        [2.0551, 0.8105, 0.8767],
        [0.6735, 0.2657, 0.2873],
        [2.4355, 0.9606, 1.0390],
        [3.2192, 1.2697, 1.3733],
        [1.4357, 0.5663, 0.6125],
        [0.9341, 0.3684, 0.3985]], device='cuda:0')
```