PyTorch_Basics_Tutorial_final

August 19, 2023

[1]:

```
import numpy as np
print(np.__version__)
import torch
print(torch.__version__)
import matplotlib.pylab as plt
```

```
1.23.5
2.0.1+cu118
```

Correlation: PyTorch vs Numpy

[2]:

```
# Create a numpy array of shape (2,3) and print its shape
numpy_array = np.array([[1,2,3], [4,5,6]])
print(numpy_array.shape)

# create a tensor of shape (2,3) and print its shape
torch_tensor = torch.tensor([[1,2,3,], [4,5,6]])
print(torch_tensor.shape)
```

```
(2, 3)
torch.Size([2, 3])
```

[3]:

```
# Generate a random number of shape (3,4) in numpy
numpy_rand = np.random.rand(3,4)
print(numpy_rand)

# Generate a random number of shape (3,4) in PyTorch
torch_rand = torch.rand(3,4)
print(torch_rand)
```

```
[[0.16217428 0.82824196 0.54467197 0.41789477]
 [0.26921656 0.41767813 0.2164388  0.09966194]
 [0.19866075 0.13359389 0.38171415 0.48985428]]
tensor([[0.3064, 0.4763, 0.5908, 0.6238],
        [0.8397, 0.1822, 0.4500, 0.2720],
        [0.0799, 0.1624, 0.8114, 0.2577]])
```

[4]:

```
# Generate zeros of shape (1,10) in numpy
numpy_zeros = np.zeros((1,10))
print(numpy_zeros)

# Generate zeros of shape (1,10) in torch
torch_zeros = torch.zeros((1,10))
print(torch_zeros)
```

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

[5]:

```
# Generate ones of shape (1,7) in numpy
numpy_ones = np.ones((1,7))
print(numpy_ones)

# Generate ones of shape (1,7) in torch
torch_ones = torch.ones((1,7))
print(torch_ones)
```

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

[6]:

```
# create a range of values 0 to 10 in numpy

zero_to_ten_np = np.arange(0,10)
print(zero_to_ten_np)

# Create a range of values 0 to 10 in torch
zero_to_ten_torch = torch.arange(0,10)
print(zero_to_ten_torch)
```

```
[0 1 2 3 4 5 6 7 8 9]
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

List/Array/Tensor Manipulation

[7]:

```
int_list = [1,2,3,4,5]
# Convert a integer list with length 5 to a tensor
int_tensor = torch.tensor(int_list)
print(int_tensor.dtype)
```

```
torch.int64
```

[8]:

```
# Convert a float list with length 5 to a tensor
float_list = [0.0, 1.0, 2.0, 3.0, 4.0]
# YOUR CODE STARTS HERE
floats_to_tensor = torch.tensor(float_list)
print(floats_to_tensor)
print(floats_to_tensor.dtype)
#YOUR CODE ENDS HERE
```

```
tensor([0., 1., 2., 3., 4.])
torch.float32
```

[9]:

```
# Convert the integer list to float tensor
old_int_tensor = torch.tensor([0, 1, 2, 3, 4])
# YOUR CODE STARTS HERE
new_float_tensor = old_int_tensor.float()
print(new_float_tensor)
#YOUR CODE ENDS HERE
```

```
tensor([0., 1., 2., 3., 4.])
```

[9]:

```

```

numpy vs. torch * Convert the given numpy array to a torch tensor; And torch tensor to a numpy array

[10]:

```
twoD_list = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
twoD_numpy = np.asarray(twoD_list)
print("The numpy array: ", twoD_numpy)
print("Type : ", twoD_numpy.dtype)

# Convert numpy array to tensor
# YOUR CODE STARTS HERE
twoD_tensor = torch.tensor(twoD_numpy,dtype=float)
print(twoD_tensor)
print(twoD_tensor.shape)
twoD_tensor = torch.asarray(twoD_numpy,dtype=float)
print(twoD_tensor)
print(twoD_tensor.shape)

#YOUR CODE ENDS HERE
print("\nNumpy Array -> Tensor:")
print("The tensor after converting:", twoD_tensor)
print("Type after converting: ", twoD_tensor.dtype)

# Convert torch tensor to numpy array
# YOUR CODE STARTS HERE

print("\n\n")
new_twoD_numpy = twoD_tensor.numpy()
print(twoD_tensor.shape)

#YOUR CODE ENDS HERE
print("\nTensor -> Numpy Array:")
print("The numpy array after converting: ", new_twoD_numpy)
print("Type after converting: ", new_twoD_numpy.dtype)
```

```
The numpy array:  [[11 12 13]
 [21 22 23]
 [31 32 33]]
Type :  int64
tensor([[11., 12., 13.],
        [21., 22., 23.],
        [31., 32., 33.]], dtype=torch.float64)
torch.Size([3, 3])
tensor([[11., 12., 13.],
        [21., 22., 23.],
        [31., 32., 33.]], dtype=torch.float64)
torch.Size([3, 3])

Numpy Array -> Tensor:
The tensor after converting: tensor([[11., 12., 13.],
        [21., 22., 23.],
        [31., 32., 33.]], dtype=torch.float64)
Type after converting:  torch.float64



torch.Size([3, 3])

Tensor -> Numpy Array:
The numpy array after converting:  [[11. 12. 13.]
 [21. 22. 23.]
 [31. 32. 33.]]
Type after converting:  float64
```

[10]:

```

```

2D Torch Tensors and 2D numpy arrays

Access the different elements of the tensor `twoD_tensor` and numpyarray `twoD_numpy`.

[11]:

```
# Slice rows 2nd and 3rd row
# YOUR CODE STARTS HERE
sliced_tensor = twoD_tensor[1:3, :]
sliced_numpy = twoD_numpy[1:3, :]
# YOUR CODE STARTS HERE

print("Tensor: Result after tensor slicing ", sliced_tensor)
print("Tensor: Dimension after tensor slicing ", sliced_tensor.ndimension())
print("Numpy: Result after np slicing: ", sliced_numpy)
print("Numpy: Dimension after np slicing: ", sliced_numpy.ndim)
```

```
Tensor: Result after tensor slicing  tensor([[21., 22., 23.],
        [31., 32., 33.]], dtype=torch.float64)
Tensor: Dimension after tensor slicing  2
Numpy: Result after np slicing:  [[21 22 23]
 [31 32 33]]
Numpy: Dimension after np slicing:  2
```

[11]:

```python

```

Dot Product

In this task, you will implement the dot product function for numpy arrays & torch tensors.

The dot product (also known as the scalar product or inner product) is the linear combination of the n real components of two vectors.

$$x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

Your Task: Implement the functions `NUMPY_dot` & `PYTORCH_dot`.

[12]:

```python
def NUMPY_dot(x, y):
    """
    Dot product of two arrays.

    Parameters:
    x (numpy.ndarray): 1-dimensional numpy array.
    y (numpy.ndarray): 1-dimensional numpy array.

    Returns:
    numpy.int64: scalar quantity.
    """
    # YOUR CODE STARTS HERE

    out = np.dot(x,y)

    # YOUR CODE ends HERE

    return out
```

[13]:

```python
def PYTORCH_dot(x, y):
    """
    Dot product of two tensors.

    Parameters:
    x (torch.Tensor): 1-dimensional torch tensor.
    y (torch.Tensor): 1-dimensional torch tensor.

    Returns:
```

```
    torch.int64: scalar quantity.
    """
    # YOUR CODE STARTS HERE

    out = torch.dot(x,y)

    # YOUR CODE ends HERE

    return out
```

[14]:
```
# TEST cases
X = np.asarray([1,2,3])
Y = np.asarray([4,-5,6])
print(f'NUMPY: Dot product of {X} and {Y} is {NUMPY_dot(X,Y)}')
assert NUMPY_dot(X,Y)==12

X = torch.from_numpy(X)
Y = torch.from_numpy(Y)
print(f'Pytorch: Dot product of {X} and {Y} is {PYTORCH_dot(X,Y)}')
assert PYTORCH_dot(X,Y).item()==12
```

```
NUMPY: Dot product of [1 2 3] and [ 4 -5  6] is 12
Pytorch: Dot product of tensor([1, 2, 3]) and tensor([ 4, -5,  6]) is 12
```

Creating a Tensor & Understanding it

[15]:
```
tensor = torch.tensor([[[1, 2, 3],
                        [3, 6, 9],
                        [2, 4, 5]]])
tensor
```

[15]:
```
tensor([[[1, 2, 3],
         [3, 6, 9],
         [2, 4, 5]]])
```

[16]:
```
# print the shape of the above tensor
print(tensor.shape)
```

```
torch.Size([1, 3, 3])
```

[17]:
```
## Can you correleate it with (batch_size, channels, height, width)?
```

Assuming that we are dealing with a grayscale image (Only 3 dimensions available in the vector, instead of 4), I'd say that batch_size is 1, height and width are 3 units each. And obviously, there is 1 channel.

[17]:
```

```

Tensor Datatypes

[18]:

```
# Default datatype for tensors is float32
float_32_tensor = torch.tensor([3.0, 6.0, 9.0],
                               dtype=None, # defaults to None, which is torch.
 ↪float32 or whatever datatype is passed
                               device=None, # defaults to None, which uses the␣
 ↪default tensor type
                               requires_grad=False) # if True, operations␣
 ↪performed on the tensor are recorded

float_32_tensor.shape, float_32_tensor.dtype, float_32_tensor.device
```

[18]:

```
(torch.Size([3]), torch.float32, device(type='cpu'))
```

[18]:

```

```

Getting information from tensors

[19]:

```
# Create a tensor
some_tensor = torch.rand(3, 4)

# Find out details about it
print(some_tensor)
print(f"Shape of tensor: {some_tensor.shape}")
print(f"Datatype of tensor: {some_tensor.dtype}")
print(f"Device tensor is stored on: {some_tensor.device}") # will default to CPU
```

```
tensor([[4.8862e-01, 9.3175e-01, 3.6153e-01, 3.8928e-04],
        [8.1481e-01, 4.4611e-01, 2.0216e-01, 1.5740e-01],
        [9.6129e-01, 5.6293e-01, 8.4194e-01, 4.4290e-01]])
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

Common Errors * Data type mismatch * Shape mismatch * Variable device mismatch

Basics tensor operations

[20]:

```
tensor = torch.tensor([1, 2, 3])
# multiply tensor by 20
tensor_mul = torch.mul(tensor, 20)
print(tensor_mul)

# add 13 to each element of the tensor
tensor_add = torch.add(tensor, 13)
print(tensor_add)
```

```
tensor([20, 40, 60])
tensor([14, 15, 16])
```

[21]:

```
#built-in functions like torch.mul() (short for multiplication) and torch.add() to perform basic operations.
#torch.mm() which is a short for torch.matmul()
```

[22]:

```
matrix1 = torch.tensor([[1, 2],
                        [3, 4]])
matrix2 = torch.tensor([[5, 6],
                        [7, 8]])
tensor_mm = torch.mm(matrix1, matrix2)
print(tensor_mm)

tensor_matmul = torch.matmul(matrix1, matrix2)
print(tensor_matmul)

print(tensor_mm.all() == tensor_matmul.all())
```

```
tensor([[19, 22],
        [43, 50]])
tensor([[19, 22],
        [43, 50]])
tensor(True)
```

Change tensor datatype

[23]:

```
# Create a tensor and check its datatype
tensor = torch.arange(10., 100., 10.)
tensor.dtype
```

[23]:

```
torch.float32
```

[24]:

```
# Create a float16 tensor
tensor_float16 = tensor.type(torch.float16)
tensor_float16
```

[24]:

```
tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.], dtype=torch.float16)
```

GPU

[25]:

```
import torch
!nvidia-smi
```

```
Sat Aug 19 17:34:55 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P8     9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[26]:
```python
torch.cuda.is_available()
```

[26]:
```
True
```

[27]:
```python
# Set device type
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

[27]:
```
'cuda'
```

[28]:
```python
torch.cuda.device_count()
```

[28]:
```
1
```

[29]:
```python
# Create tensor (default on CPU)
tensor = torch.tensor([1, 2, 3])

# Tensor not on GPU
print(tensor, tensor.device)

# Move tensor to GPU (if available)
tensor_on_gpu = tensor.to(device)
tensor_on_gpu
```

```
tensor([1, 2, 3]) cpu
```

[29]:
```
tensor([1, 2, 3], device='cuda:0')
```

[30]:
```python
# copy the tensor back to cpu
tensor_back_on_cpu = tensor_on_gpu.cpu().numpy()
tensor_back_on_cpu
```

[30]:
```
array([1, 2, 3])
```

** Computer Vision/ Imaging Related Pytorch library**

Image credit: https://www.learnpytorch.io/03_pytorch_computer_vision/

[30]: