PYTORCH

* Pytorch :- It is an open-source machine learning library developed by facebook AI it is used to build deep learning models and perform numerical computations using python

It is easy and flexible to build and train neural network
The core of AI systems like:-
- Image recognition
- Natural language processing (NLP)
- Recommendation system
- Generative AI (eg., chatGPT, DALL-E)

* Tensors:-
A Tensor is just a multi-dimensional array, like a Numpy array, but with extra features that make it powerful for deep learning.
- 0D Tensor :- a single number -> scalar (1, 2, 0.2, 10, ...)
- 1D tensor :- a list of numbers -> vector
- 2D tensor :- a matrix (rows & columns)
- 3D+ tensors :- used for images, video, batches of data,

* creating a Tensor :-
```
import torch
a = torch.tensor([7, 4, 3, 2, 67])
```
a[0]:7, a[1]:4, a[2]:3, a[3]:2, a[4]:6

a.dtype → (To find the type of data which is stored in the tensor)
torch.int64
a.type()
torch.LongTensor

```
Import torch
a=torch.tenpor([0.0,1.0,2.0,3.0,0.4])
a.dtype
torch.Float32
a.type
```

- we can also specify the data type using parameter d type:-

```
Import torch
a=torch.tenpor([0.1,0.0,0.2,0.3,4.0], dtype=
        torch.Int32)
```

- we can also create a float tenpor explicitly using the Float Tenpor method:-

```
Import torch
a=torch.FloatTenpor([0,1,2,3,4])
a.type()->(to check which type)
torch.FloatTenpor
a.tenpor[(0.,1.,2.,3.,4.)]
```

(When we print out a, we can see the decimals were added to the numbers in the tenpor)

- we can convert the type of the tenpor to float using the type method, passing in the argument torch.Float Tenpor:-

```
Import torch
a=torch.tenpor([0,1,2,3,4])
a=a.type(torch.FloatTenpor)
a.Size(): torch.Size(5)
a.ndimension(): 1
```

In numpy (or even in regular Python arrays):-
a_col = a.reshape(6,1)

In Pytorch:-
a_col = a.view(-1,1)
(Figure out how many rows are needed, I only care that it has 1 column.)

- Con... a 2-D tensor using the view...

Imp...
a = t... ,])
a_col = a.... (we to know exact shape)
a_col = a.view(-1,1) -> (Figure out how many rows needed)
(-1 allows pytorch to infer the number of rows in the new tensor, rather than explicitly setting it

we can convert a numpy array to a torch tensor using the function "from numpy".
&
then we can also convert the tensor back to numpy array using the method numpy.
&
we can convert a pandas series to a tensor in similar manner.

...h
...y = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
...] = torch.from_numpy(numpy_array)
...numpy = torch_tensor.numpy()

import torch
Pandas_series = pd.Series([0.1, 2.0, 3.0, 4.0])
Pandas_to_torch = torch.from_numpy(Pandas_series.values)
(we simply use the attribute "values" to convert the series to a numpy array. we then use the function from numpy to convert it to tensor.

- The method "to_list" to return a list from a tensor
this_tensor = torch.tensor([0,1,2,3])
torch_to_list = this_tensor.tolist()
torch_to_list : [0,1,2,3]

convert these tensor into a 2-D tensor using the view method in Pytorch.

```
import torch
a = torch.Tensor([1,2,3,4,5])
a_col = a.view(5,1) -> (U have to know exact shape)
a_col = a.view(-1,1) -> (Figure out how many rows needed)
```

(-1 allows pytorch to infer the number of rows in the new tensor, rather than explicitly setting it to "1".

```
import torch
numpy_array = np.array([0.0,1.0,2.0,3.0,4.0])
torch_tensor = torch.from_numpy(numpy_array)
back_to_numpy = torch_tensor.numpy()
```

```
import torch
Pandas_series = pd.Series([0.1,2.0,3.0,4.0])
Pandas_to_torch = torch.from_numpy(Pandas_series.
                  values)
```

(we simply use the attribute "values" to convert the series to a numpy array. we then use the function from numpy to convert it to tensor.

The method "to_list" to return a list from a tensor

```
this_tensor = torch.tensor([0,1,2,3])
torch_to_list = this_tensor.tolist()
torch_to_list : [0,1,2,3]
```

- Individual value of a tensor are also tensors
new_tensor = torch.tensor([5,2,6,1])
new_tensor[0]: tensor(5)
new_tensor[1]: tensor(2)
new_tensor[2]: tensor(6)
new_tensor[3]: tensor(1)

If we work a python number instead of tensors for these we use the method "item" to return a number.
new_tensor[0].item(): 5
new_tensor[1].item(): 7
new_tensor[2].item(): 6
new_tensor[3].item(): 1

* Indexing & slicing:-
A method that you can use to access a particular value or set of values stored in the tensor.
Eg:-
C = torch.tensor([20,1,2,3,4])
C: tensor([20,1,2,3,4])
C[0]=100
C: tensor([100,1,2,3,4])
C[4]=0
C: tensor([100,1,2,3,0])
Slicing -
C: tensor([100,1,2,3,0])

0	1	2	3	4

d=C[1:4]
d: tensor([1,2,3])

C : tensor([100, 1, 2, 3, 0])
[0] [1] [2] [3] [4]

C[3:5] = torch.tensor([300.0, 400.0])

C : tensor([100, 1, 2, 300, 400])

* Basic operations :-

- vector addition and subtraction :-

$$u = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad v = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$z = u + v = \begin{bmatrix} 1+0 \\ 0+1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

(new vector z is now a linear combination of the vector 'u' and 'v')

In Pytorch :-

```
u = torch.tensor([1.0, 0.0])
v = torch.tensor([0.0, 1.0])
z = u + v
z : tensor([1, 1])
```

- vector multiplication with a scalar :-

$$Y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$z = 2Y = \begin{bmatrix} 2(1) \\ 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

In Pytorch :-

```
Y = torch.tensor([2, 1])
Z = 2 * Y
Z : tensor([2, 4])
```

- Product of two tensors:-

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad v = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$z = u \cdot v = \begin{bmatrix} 1*3 \\ 2*2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

In Pytorch:-

```
u = torch.tensor([1,2])
v = torch.tensor([3,2])
z = u*v
z: tensor([3,4])
```

- Dot Product:-

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$u^T v = 1 \times 3 + 2 \times 1$$

$$= 5$$

In Pytorch:-

```
u = torch.tensor([1,2])
v = torch.tensor([2,1])
result = torch.dot(u,v)
result: 5
```

- Adding constant to a tensor:-

```
u = torch.tensor([1,2,3,-1])
z = u+1
z: tensor([2,3,4,0])
```

* Functions :-

• universal function :-

```
a=torch.tensor([1,-1,1,-1])
mean_a=a.mean()
 mean_a: 0.0
```

$\frac{1}{4}(1-1+1-1)$

$= 0$

```
b=torch.tensor([1,-2,3,4,5])
max_b=b.max()
 max_b: 5
```

```
np.pi
x=torch.tensor([0,np.pi/
                2,np.pi])
y=torch.sin(x)
```

$x=\left[0, \frac{\pi}{2}, \pi\right]$

$y=\left[\sin(0), \sin\left(\frac{\pi}{2}\right), \sin(\pi)\right]$

```
y: tensor([0,1,0])
```

$y=[0,1,0]$

torch.linspace(-2, 2, steps=5)
(start point) (end point)

(It returns evenly spaced numbers over a specified interval)

-2	-1	0	1	2

torch.linspace(-2, 2, steps=9)

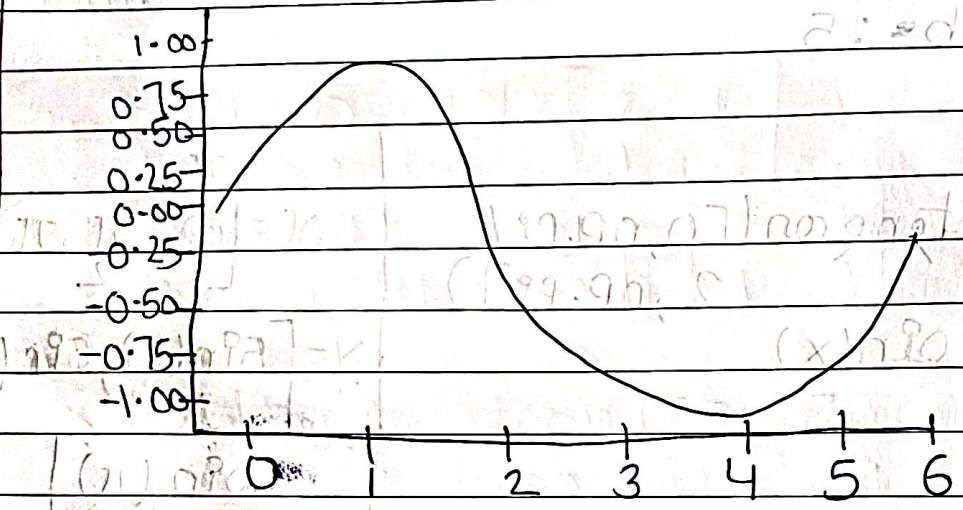
-2	-1.5	-1	-0.5	0	0.5	1	1.5	2

• Plotting Mathematical functions:-

```
x = torch.linespace(0, 2*np.pi, 100)
y = torch.sin(x)

import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(x.numpy(), y.numpy())
```

↳(convert x into numpy array similar to y)



* Two dimensional Tensors:-

• Tensor creation in 2D:-

```
a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
A = torch.tensor(a)
```

A:

11	12	13
21	22	23
31	32	33

```
A.ndimension(): 2
A.shape : torch.Size(3,3)
A.size(): (3,3)
A.numel(): 9 → (To obtain the no of elements in a tensor)
```

classmate
Date
Page

• Indexing and Slicing in 2D:-

A: [[A[0][0], A[0][1], A[0][2]], [A[1][0], A[1][1], A[1][2]] [A[2][0], A[2][1], A[2][2]]]

→ row Index → column Index

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]

A=[[11,12,13],[21,22,23],[31,32,33]]

A[1][2]

row \ column	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

A[0][0]

11	12	13
21	22	23
31	32	33

Slicing:- A[0,0:2] : ([11, 12])

row \ column	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

A[1:3, 2] : ([23, 33])

	0	1	2
0	11	12	13
1	21	22	(23)
2	31	32	(33)

* Basic operations in 2D

• Adding 2 tensors:- (matrix addition)

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X+Y = \begin{bmatrix} 1+2 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} \quad \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

In Pytorch:-

```
X = torch.tensor([[1,0],[0,1]])
Y = torch.tensor([[2,1],[1,2]])
Z = X + Y;
Z : tensor([[3,1],[1,3]])
```

• Multiplying tensor by a scalar:-

$$Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$2Y = \begin{bmatrix} 2\times2 & 1\times2 \\ 2\times1 & 2\times2 \end{bmatrix} \quad \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

In Pytorch:-

```
Y = torch.tensor([[2,1],[1,2]])
Z = 2 * Y;
Z : tensor([[4,2],[2,4]])
```

classmate
Date
Page

- Multiplying 2 tensors :-

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X \cdot Y = \begin{bmatrix} (1)2 & (0)1 \\ (0)1 & (1)2 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

In Pytorch :-

```
X = torch.tensor([[1,0],[0,1]])
Y = torch.tensor([[2,1],[1,2]])
Z = X * Y;
Z : tensor([[2,0],[0,2]])
```

- Matrix Multiplication :-

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$0 \times 1 + 1 \times 1 + 1 \times (-1) = 0$

$$AB = \begin{bmatrix} 0 & 2 \\ 0 & 2 \end{bmatrix}$$

In Pytorch :-

```
A = torch.tensor([[0,1,1],[1,0,1]])
B = torch.tensor([[1,1],[1,1],[-1,1]])
C = torch.mm(A,B); ->(mm method is used for performing matrix multiplication)
C : tensor([[0,2],[0,2]])
```

Y.backward():-
It first calculates the derivative of Y and then evaluates it for the value of x equals 2.

* Differentiat

• Derivatives

$Y(x) = x^2$

$Y(2) = 2^2$

$\frac{dY(x)}{d(x)} = 2x$

$\frac{dY(2)}{dx} = 2(2) = 4$

In Pytorch:-

x = torch.tensor(2, requires_grad=True) -> Y(x)=

Y = x**2 -> Y(x=2) = 2^2

Y.backward() -> $\frac{dY(x=2)}{dx} = 2x$ (we are telling pytorch th we would be using eval ting fn & derivatives of x using this value of x)

x.grad -> $\frac{dY(2)}{dx} = 2(2) = 4$

$Z(x) = x^2 + 2x + 1$

$Z(2) = 2^2 + 2(2) + 1 = 9$

$\frac{dz(x)}{dx} = 2x + 2$

$\frac{dz(2)}{dx} = 2(2) + 2$

$= 6$

In Pytorch:-

```
x = torch.tensor(2, requires_grad=True)
z = x**2 + 2*x + 1
z.backward()
x.grad
6
```

* Differentiation in Pytorch:-

- Derivatives:-

$Y(x) = x^2$

$Y(2) = 2^2 = 4$

$\frac{dY(x)}{d(x)} = 2x^1$

$\frac{dY(2)}{dx} = 2(2) = 4$

In Pytorch:-

x = torch.tensor(2, required (grad=True)) -> Y(x)

Y = x**2 -> Y(x=2) = 2^2

Y.backward() -> $\frac{dY(x=2)}{dx} = 2x$ (we are telling pytorch that we would be using any fn & derivatives of using this value of x)

x.grad -> $\frac{dY(2)}{dx} = 2(2) = 4$

$Z(x) = x^2 + 2x + 1$

$Z(2) = 2^2 + 2(2) + 1 = 9$

$\frac{dZ(x)}{dx} = 2x + 2$

$\frac{dZ(2)}{dx} = 2(2) + 2$

$= 6$

In Pytorch:-

```
x = torch.tensor(2, required_grad=True)
z = x**2 + 2*x + 1
z.backward()
x.grad
6
```

classmate
Date
Page

- Partial derivatives:-

$$f(u,v) = uv + u^2$$

$$\frac{\partial f(u,v)}{\partial u} = v + 2u \qquad \frac{\partial f(u,v)}{dv} = u$$

In Pytorch:-

```
u = torch.tensor(1, requires_grad=True)
v = torch.tensor(2, requires_grad=True)
f = u*v + u**2
f.backward()
u.grad
tensor(4)
```

calculating the derivative of f with respect to v

```
-11-
-11-
-11-
-11-
v.grad
tensor(1)
```

* Simple dataset :-

```
From torch.utils.data import Dataset
class toy_set (Dataset):
    def __init__(self, length=100, transform = None):
        self.x=2* torch.ones(length,2)
        self.y= torch.ones(length,1)
        self.len=length
        self.transform= transform

    def __getitem__(self,index):
        sample = self.x[index], self.y[index]
        if self.transform:
            sample = self.transform(sample)
        return sample

    def __len__(self):
        return self.len
```

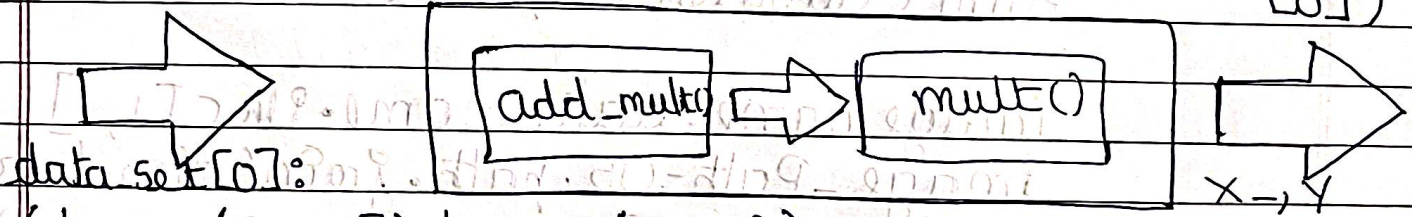
- Transforms :- (To transform the data for example normalize or standardize the data.)

```
class add_mult(object):
def __init__(self, addx=1, muly=1):
    self.addx=addx
    self.muly= muly
def __call__(self, sample):
    x= sample[0]
    y= sample[1]
    x= x+self.addx
    y= y*self.muly
    sample=x,y
    return sample
```

- Transforms compose:- (run several transforms in series.)

```
class mult(object):
    def __init__(self, mul=100):
        self.mul = mul
    def __call__(self, sample):
        x = sample[0]
        y = sample[1]
        x = x * self.mul
        y = y * self.mul
        sample = x, y
        return sample
```

```
from torchvision import transforms
data_transform = transforms.Compose([add_mult(),
                                     mult()])
x_, y_ = data_transform(data_set[0])
```



data_set[0]:
(tensor([2., 2.]), tensor([1.]))

x_, y_
(tensor([300., 300.]), tensor([200.]))

Dataset used (Zalanda's Fashion MNIST) → 60,000 Im

* Data Set:- (Images)

Data set class:

```
From PIL import Image
import Pandas as pd
import OS
from matplot lib. pyplot import imshow
from torch. utils. data import Dataset,
                               Dataloader

Directory = "/resources/data"
CSV_File = 'Index. CSV'
CSV_Path = OS. path. join (directory, CSV_Fil

data_name = pd. read_CSV (CSV_Path)
data_name. head()

Print ('File name:', data_name. iloc [1,1])
Print ('class on Y:', data_name. iloc [1,0])

Image name = data_name. iloc [1,1]
Image_Path = OS. path. join (directory,
                             image_name)

Image = Image. open (Image_Path)
plt. imshow (Image, cmap='gray', vmin=0,
                        vmax=255)
plt. title (data_name. iloc [1,0])
plt. show()
```

```
class Dataset(Dataset):
    def __init__(self, csv_file, data_dir,
                 transform=None):
        self.transform = transform
        self.data_dir = data_dir
        data_dir_csv_file = os.path.join(self.
            data_dir, csv_file)
        self.data_name = pd.read_csv(data_
            dir_csv_file)
        self.len = self.data_name.shape[0]

    def __len__(self):
        return self.len
    def __getitem__(self, idx):
        img_name = os.path.join(self.data_
            name.iloc[idx, 1]
        image = Image.open(image_name)
        Y = self.data_name.iloc[idx, 0]
        if self.transform:
            image = self.transform(image)
        return image, Y
```

- Torch vision transforms :-

```
import torchvision.transforms as transfo
                                rms
transforms.CenterCrop(20)
transforms.ToTensor()
self.data_transform = transforms.
    Compose([transforms.CenterCrop
    (20), transforms.ToTensor()])
```

dataset = Dataset(CSV_file=CSV_File, data_dir=directory, transform=
custom_data_transform)
dataset[0][0].shape
torch.Size([1, 20, 20])

- Torchvision Datasets:- (Pre-built datasets to compare models)

Import torchvision.datasets as dsets
dataset = dsets.MNIST(root='./data', train=False, download=True, transform=transforms.ToTensor())