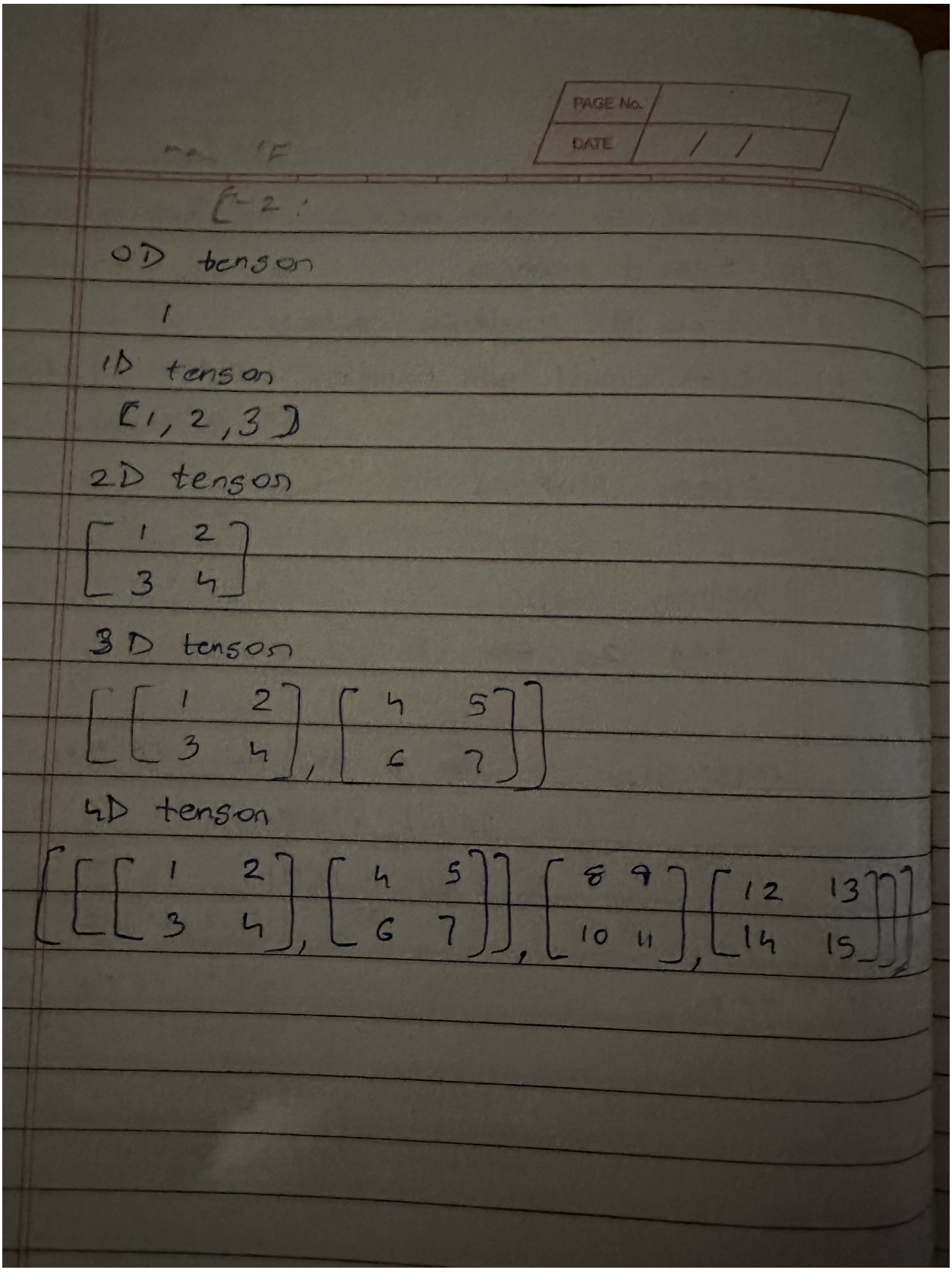
Perceptron and Linear Algebra for Deep Learning

> Assignment 4

1. **Create one example each for a OD Tensor, 1D Tensor, 2D Tensor, 3D Tensor, and a 4D Tensor. This is to be done (on paper or document) without writing any code, imagining tensors with arbitrary values in them.**


PAGE No.
DATE
/ /
0D tenson
1
1D tenson
[1, 2, 3]
2D tenson
[[1 2]
[3 4]]
3D tenson
[[[1 2]
[3 4]], [[4 5]
[6 7]]]
4D tenson
[[[[1 2]
[3 4]], [[4 5]
[6 7]]], [[[8 9]
[10 11]], [[12 13]
[14 15]]]]

2. **Create the above examples in PyTorch. Make sure that your examples are FloatTensors. Print the data type for each example to make sure they are of desired FloatTensor type. For every tensor, print the shape of the tensor to convince yourself that the tensor is of a desired rank.**

```
In [1]: import torch
        tensor_0d = torch.FloatTensor([1.0])
        print(tensor_0d.squeeze())
        print(tensor_0d.shape)
        print(tensor_0d.dtype)
```

```
tensor(1.)
torch.Size([1])
torch.float32
```

```
In [2]: tensor_1d = torch.FloatTensor([1.0, 2.0, 3.0])
        print(tensor_1d.squeeze())
        print(tensor_1d.shape)
        print(tensor_1d.dtype)
```

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

```
In [3]: tensor_2d = torch.FloatTensor([[1.0, 2.0], [3.0, 4.0]])
        tensor_2d.squeeze()
        print(tensor_2d.shape)
        print(tensor_2d.dtype)
```

```
torch.Size([2, 2])
torch.float32
```

```
In [4]: tensor_3d = torch.FloatTensor([[[1.0, 2.0], [3.0, 4.0]], [[4.0, 5.0], [6.0, 7.0]]])
        print(tensor_3d.squeeze())
        print(tensor_3d.shape)
        print(tensor_3d.dtype)
```

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

        [[4., 5.],
         [6., 7.]]])
torch.Size([2, 2, 2])
torch.float32
```

```
In [5]: tensor_4d = torch.FloatTensor([[[[1.0, 2.0], [3.0, 4.0]], [[4.0, 5.0], [6.0, 7.0]]], [[[8.0, 9.0], [10.0, 11.0]], [[12.0, 13.0], [14.0, 15.0]]]])
        print(tensor_4d.squeeze())
        print(tensor_4d.shape)
        print(tensor_4d.dtype)
```

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

         [[ 4.,  5.],
          [ 6.,  7.]]],


        [[[ 8.,  9.],
          [10., 11.]],

         [[12., 13.],
          [14., 15.]]]])
torch.Size([2, 2, 2, 2])
torch.float32
```

3. **Create a version of the above tensors that are IntTensors. Print the data type to ensure that each example is of the desired IntTensor type.**

```
                [14., 15.]]]])
```

In [9]:
```
tensor_0d_int = tensor_0d.to(torch.int)
tensor_0d_int
```

Out[9]:
```
tensor([1], dtype=torch.int32)
```

In [12]:
```
tensor_1d_int = tensor_1d.to(torch.int)
tensor_1d_int
```

Out[12]:
```
tensor([1, 2, 3], dtype=torch.int32)
```

In [13]:
```
tensor_2d_int = tensor_2d.to(torch.int)
tensor_2d_int
```

Out[13]:
```
tensor([[1, 2],
        [3, 4]], dtype=torch.int32)
```

In [14]:
```
tensor_3d_int = tensor_3d.to(torch.int)
tensor_3d_int
```

Out[14]:
```
tensor([[[1, 2],
         [3, 4]],

        [[4, 5],
         [6, 7]]], dtype=torch.int32)
```

In [15]:
```
tensor_4d_int = tensor_4d.to(torch.int)
tensor_4d_int
```

Out[15]:
```
tensor([[[[ 1,  2],
          [ 3,  4]],

         [[ 4,  5],
          [ 6,  7]]],


        [[[ 8,  9],
          [10, 11]],

         [[12, 13],
          [14, 15]]]], dtype=torch.int32)
```

4. **Create two 1D Tensors of same length in PyTorch with arbitrary values. How do you determine if the angle between the two tensors is < 90 degrees or > 90 degress? Demonstrate this using code.**

5. Create two 1D Tensors of same dimensions with arbitrary values and call them `a` and `b`. Load tensor `a` to a GPU and leave tensor `b` on the CPU.
    a. Now, compute the dot product of these two tensors. What issues did you run into, if any, and why?
    b. Load tensor `b` also to the GPU and then compute the dot product. What difference did you notice compared to the previous case and why?

```
import torch

# use mac gpu
device = torch.device('mps')

a = torch.tensor([1.0, 2.0, 3.0], device=device)   # loaded on g
b = torch.tensor([4.0, 5.0, 6.0])   # loaded on cpu

try:
    product = a@b
    print(product)
```

```
except Exception as e:
    print(f"Error >> {e}")

"""
Returns
Expected all tensors to be on the same device. Found: cpu, mps:0
"""

b = b.to(device)

try:
    product = a@b
    print(product)
except Exception as e:
    print(f"Error >> {e}")

"""
Returns
tensor(32., device='mps:0')

"""
```

torch_with_gpu.py

```python
import torch

# use mac gpu
device = torch.device('mps')

a = torch.tensor( data: [1.0, 2.0, 3.0], device=device)  # loaded on gpu
b = torch.tensor([4.0, 5.0, 6.0])  # loaded on cpu

try:
    product = a @ b
    print(product)
except Exception as e:
    print(f"Error >> {e}")

"""
Returns
Expected all tensors to be on the same device. Found: cpu, mps:0
"""

b = b.to(device)

try:
    product = a @ b
    print(product)
except Exception as e:
    print(f"Error >> {e}")
```

```
Terminal  Local ×  +
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+ python L04_linalg-dl/homework/torch_with_gpu.py
Error >> Expected all tensors to be on the same device. Found: mps:0, cpu
tensor(32., device='mps:0')
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+
```

case (a) fails because while performing a dot product pytorch requires the two respective tensors to be loaded on the same device (hardware).

case (b) works because both the tensors are loaded on the same device (hardware) i.e. gpu