

Assignment 02 (St ID: 1174066)

OUTPUTS

MatrixOps.py

```
Run MatrixOps x
"E:\2.Online MS CS\2nd Year\repo\dev-cs\CPSC_592_NLP_LLM\Assignments\Assignment_02\
Printing tensor a:
  tensor([[0, 1],
         [2, 3]])
Printing tensor b:
  tensor([[0, 1],
         [2, 3]])
Printing element by element multiplication:
  tensor([[0, 1],
         [4, 9]])
Printing matrix addition: tensor([[0, 2],
         [4, 6]])
Printing tensor e , 2 rows and 3 columns values ranging from 0 to 5:
  tensor([[0, 1, 2],
         [3, 4, 5]])
Printing matrix multiplication: tensor([[ 3,  4,  5],
         [ 9, 14, 19]])
Printing matrix multiplication (alternative syntax):
  tensor([[ 3,  4,  5],
         [ 9, 14, 19]])
Printing transpose, dim swapped:
  tensor([[ 3,  9],
         [ 4, 14],
         [ 5, 19]])
Printing tensor f with an added dimension in the beginning:
torch.Size([1, 2, 3])
f1 is: tensor([[[ 3,  4,  5],
                 [ 9, 14, 19]])])
```

Run

MatrixOps x



Printing transpose, dim swapped.

```
tensor([[ 3,  9],
        [ 4, 14],
        [ 5, 19]])
```

Printing tensor f with an added dimension in the beginning:

```
torch.Size([1, 2, 3])
f1 is: tensor([[[ 3,  4,  5],
                [ 9, 14, 19]]])
```

Printing 2x2x3 tensor matrix:

```
tensor([[[ 1,  2,  3],
         [ 4,  5,  6]],
        [[ 7,  8,  9],
         [10, 11, 12]]]) and
```

size: torch.Size([2, 2, 3])

Printing transpose with the added dimension:

```
tensor([[[ 3,  9],
         [ 4, 14],
         [ 5, 19]]])
```

Printing batch matrix multiplication:

```
torch.Size([10, 3, 5])
```

Printing tensor y:

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

size: torch.Size([3, 3])

Printing tensor z:

```
tensor([[5, 5],
        [5, 5]])
```

Process finished with exit code 0

EinopsTest.py

```
Run EinopsTest x
"E:\2.0nline MS CS\2nd Year\repo\dev-cs\CPSC_592_NLP_LLM\Assignments\Assignment_

Printing tensor A:
tensor([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])

Printing tensor B:
tensor([[1, 2, 1, 1],
        [3, 4, 2, 5],
        [1, 3, 6, 7],
        [1, 4, 6, 8]])

Printing matrix multiplication using Einstein summation:
tensor([[ 14,  35,  47,  64],
        [ 38,  87, 107, 148],
        [ 62, 139, 167, 232],
        [ 86, 191, 227, 316]])

Printing matrix multiplication Einstein summation C1:
tensor([[ 14,  35,  47,  64],
        [ 38,  87, 107, 148],
        [ 62, 139, 167, 232],
        [ 86, 191, 227, 316]])

Printing Ax(transpose(B) - matrix mult C2:
tensor([[ 12,  37,  53,  59],
        [ 32,  93, 121, 135],
        [ 52, 149, 189, 211],
        [ 72, 205, 257, 287]])

Printing A transposed using Einstein summation:
tensor([[ 1,  5,  9, 13],
        [ 2,  6, 10, 14],
        [ 3,  7, 11, 15],
        [ 4,  8, 12, 16]])
```

Run EinopsTest x



```
tensor([ 1,  6, 11, 16])
```

Printing sum diagonal elements - trace C4:

```
tensor(34)
```

Printing sum column elements (row wise sum) C5:

```
tensor([28, 32, 36, 40])
```

Printing element wise product C6:

```
tensor([[ 1,  4,  3,  4],
        [15, 24, 14, 40],
        [ 9, 30, 66, 84],
        [13, 56, 90, 128]])
```

Printing cube elements C7:

```
tensor([[ 1,  8, 27, 64],
        [125, 216, 343, 512],
        [729, 1000, 1331, 1728],
        [2197, 2744, 3375, 4096]])
```

Printing transpose C8:

```
tensor([[ 1,  5,  9, 13],
        [ 2,  6, 10, 14],
        [ 3,  7, 11, 15],
        [ 4,  8, 12, 16]])
```

Printing multiply row wise and add each row C9:

```
tensor([ 12,  93, 189, 287])
```

Printing outer product douter:

```
tensor([[ 3,  6,  9, 12],
        [ 5, 10, 15, 20],
        [ 7, 14, 21, 28],
        [ 9, 18, 27, 36]])
```

Printing inner product dinner:

```
tensor(70)
```

```
Run EinopsTest x
tensor([[[ 0, 1, 2],
          [ 3, 4, 5],
          [ 6, 7, 8],
          [ 9, 10, 11]],
        [[12, 13, 14],
          [15, 16, 17],
          [18, 19, 20],
          [21, 22, 23]]])

Printing batch tensor 2 batch_tensor_2:
tensor([[[ 0, 1, 2, 3],
          [ 4, 5, 6, 7],
          [ 8, 9, 10, 11]],
        [[12, 13, 14, 15],
          [16, 17, 18, 19],
          [20, 21, 22, 23]]])

Printing batch matrix multiplication dmul:
tensor([[[ 20, 23, 26, 29],
          [ 56, 68, 80, 92],
          [ 92, 113, 134, 155],
          [128, 158, 188, 218]],
        [[ 632, 671, 710, 749],
          [ 776, 824, 872, 920],
          [ 920, 977, 1034, 1091],
          [1064, 1130, 1196, 1262]]])

Printing tensor shape dt:
torch.Size([3, 5, 4, 6, 8, 2, 7, 9])

Printing esum - sum over dim p:
tensor([-189.4975, -278.8332, 251.8319, 118.2083, -125.8782, -98.4815,
        -11.0327, 125.0977, 262.9241])

Printing q2 shape: torch.Size([2, 64, 16, 64])
Printing q3 shape: torch.Size([128, 16, 64])
```

TransformerLayer.py

```
TransformerLayer x
:
"E:\2.0nline MS CS\2nd Year\repo\dev-cs\CPSC_592_NLP_LLM\Assignments\Assignment_02\Transf
torch.Size([4, 100, 512])
torch.Size([2, 3, 20])
```

LayerNormTest.py

```
LayerNormTest x
:
"E:\2.0nline MS CS\2nd Year\repo\dev-cs\CPSC_592_NLP_LLM\Assignments\Assignment_02\Transforme
Original numpy array: [0 1 2 3]
Standard deviation of the array: 1.118033988749895

---- Manual normalization ----
Normalized numpy array: [-1.34164079 -0.4472136  0.4472136  1.34164079]

PyTorch Tensor: tensor([0., 1., 2., 3.])

Output from the neural network:
tensor([[-1.3416, -0.4472,  0.4472,  1.3416]],
      grad_fn=<NativeLayerNormBackward0>)
```

TriuTest.py

```
run | Indirect x
Mask with upper triangle filled with ones and converted to boolean:
tensor([[ True,  True,  True,  True],
        [False,  True,  True,  True],
        [False, False,  True,  True],
        [False, False, False,  True]])

Random attention matrix (4x4) on CPU:
tensor([[0.0358, 0.9140, 0.7946, 0.7478],
        [0.1850, 0.8875, 0.6392, 0.6431],
        [0.0326, 0.7744, 0.6149, 0.8683],
        [0.9858, 0.0732, 0.2942, 0.6431]])

Attention matrix after applying the mask (masked positions set to -infinity):
tensor([[ -inf,  -inf,  -inf,  -inf],
        [0.1850,  -inf,  -inf,  -inf],
        [0.0326, 0.7744,  -inf,  -inf],
        [0.9858, 0.0732, 0.2942,  -inf]])
```

MultinomialTest.py

```
↓ Softmax probabilities after zeroing out some entries:
tensor([0.0338, 0.0000, 0.2496, 0.0338, 0.2496, 0.0918, 0.0000])

↓ Index of one top choice (sampled probabilistically):
tensor([5])

↓ Indices of top 2 choices (sampled probabilistically):
tensor([4, 2])

↓ Indices of top 3 choices (sampled probabilistically):
tensor([2, 0, 5])
```

CrossEntropyTest.py

