40216952_2023_Lab7_Ex

March 9, 2023

1 Lab 7: Self-Attention

This lab covers the following topics:

- Gain insight into the self-attention operation using the sequential MNIST example from before.
- Gain insight into positional encodings

1.1 0 Initialization

Run the code cell below to download the MNIST digits dataset:

[ ]:
```
!wget -O MNIST.tar.gz https://activeeon-public.s3.eu-west-2.amazonaws.com/
↪datasets/MNIST.new.tar.gz
!tar -zxvf MNIST.tar.gz

import torchvision
import torch
import torchvision.transforms as transforms
from torch import nn
import torch.nn.functional as F

from torch.utils.data import Subset

dataset = torchvision.datasets.MNIST('./', download=False, transform=transforms.
↪Compose([transforms.ToTensor()]), train=True)
train_indices = torch.arange(0, 10000)
train_dataset = Subset(dataset, train_indices)

dataset=torchvision.datasets.MNIST('./', download=False, transform=transforms.
↪Compose([transforms.ToTensor()]), train=False)
test_indices = torch.arange(0, 10000)
test_dataset = Subset(dataset, test_indices)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
                                          shuffle=True, num_workers=0)

test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16,
```

```
                                            shuffle=False, num_workers=0)
```

```
--2023-03-09 15:41:11--  https://activeeon-public.s3.eu-
west-2.amazonaws.com/datasets/MNIST.new.tar.gz
Resolving activeeon-public.s3.eu-west-2.amazonaws.com (activeeon-public.s3.eu-
west-2.amazonaws.com)… 3.5.245.189
Connecting to activeeon-public.s3.eu-west-2.amazonaws.com (activeeon-
public.s3.eu-west-2.amazonaws.com)|3.5.245.189|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 34812527 (33M) [application/x-gzip]
Saving to: 'MNIST.tar.gz'

     0K … … … … …  0%  552K 61s
    50K … … … … …  0%  555K 61s
   100K … … … … …  0%  580K 60s
   150K … … … … …  0%  576K 60s
   200K … … … … …  0%  566K 60s
   250K … … … … …  0%  575K 59s
   300K … … … … …  1% 1.35M 54s
   350K … … … … …  1%  902K 52s
   400K … … … … …  1%  576K 53s
   450K … … … … …  1%  577K 53s
   500K … … … … …  1%  571K 54s
   550K … … … … …  1% 10.9M 49s
   600K … … … … …  1%  574K 50s
   650K … … … … …  2%  575K 50s
   700K … … … … …  2%  575K 51s
   750K … … … … …  2%  573K 51s
   800K … … … … …  2% 1.33M 50s
   850K … … … … …  2%  930K 49s
   900K … … … … …  2%  575K 49s
   950K … … … … …  2%  563K 50s
  1000K … … … … …  3% 1.39M 48s
  1050K … … … … …  3%  912K 48s
  1100K … … … … …  3%  569K 48s
  1150K … … … … …  3%  579K 48s
  1200K … … … … …  3%  575K 49s
  1250K … … … … …  3%  691K 48s
  1300K … … … … …  3% 2.56M 47s
  1350K … … … … …  4%  574K 47s
  1400K … … … … …  4%  572K 48s
  1450K … … … … …  4%  575K 48s
  1500K … … … … …  4%  572K 48s
  1550K … … … … …  4% 1.34M 47s
  1600K … … … … …  4%  907K 47s
  1650K … … … … …  5%  576K 47s
  1700K … … … … …  5%  583K 47s
  1750K … … … … …  5% 1.32M 46s
```

```
1800K … … … … …  5%  916K 46s
1850K … … … … …  5%  572K 46s
1900K … … … … …  5%  575K 46s
1950K … … … … …  5%  572K 47s
2000K … … … … …  6% 1.28M 46s
2050K … … … … …  6%  938K 46s
2100K … … … … …  6%  575K 46s
2150K … … … … …  6%  572K 46s
2200K … … … … …  6%  568K 46s
2250K … … … … …  6%  580K 46s
2300K … … … … …  6% 1.34M 46s
2350K … … … … …  7%  904K 45s
2400K … … … … …  7%  575K 45s
2450K … … … … …  7%  573K 46s
2500K … … … … …  7%  576K 46s
2550K … … … … …  7% 14.5M 45s
2600K … … … … …  7%  575K 45s
2650K … … … … …  7%  574K 45s
2700K … … … … …  8%  574K 45s
2750K … … … … …  8% 1.31M 45s
2800K … … … … …  8%  914K 44s
2850K … … … … …  8%  575K 44s
2900K … … … … …  8%  568K 45s
2950K … … … … …  8%  578K 45s
3000K … … … … …  8%  697K 45s
3050K … … … … …  9%  960K 44s
3100K … … … … …  9%  919K 44s
3150K … … … … …  9%  573K 44s
3200K … … … … …  9%  574K 44s
3250K … … … … …  9%  574K 44s
3300K … … … … …  9% 1.29M 44s
3350K … … … … … 10%  933K 44s
3400K … … … … … 10%  582K 44s
3450K … … … … … 10%  575K 44s
3500K … … … … … 10% 1.32M 43s
3550K … … … … … 10%  915K 43s
3600K … … … … … 10%  573K 43s
3650K … … … … … 10%  575K 43s
3700K … … … … … 11%  572K 43s
3750K … … … … … 11% 1.30M 43s
3800K … … … … … 11%  601K 43s
3850K … … … … … 11%  863K 43s
3900K … … … … … 11%  576K 43s
3950K … … … … … 11%  573K 43s
4000K … … … … … 11%  574K 43s
4050K … … … … … 12% 1.32M 43s
4100K … … … … … 12%  915K 42s
4150K … … … … … 12%  570K 43s
```

```
4200K … … … … … 12%  577K 43s
4250K … … … … … 12% 1.37M 42s
4300K … … … … … 12%  917K 42s
4350K … … … … … 12%  573K 42s
4400K … … … … … 13%  576K 42s
4450K … … … … … 13%  573K 42s
4500K … … … … … 13% 1.29M 42s
4550K … … … … … 13%  927K 42s
4600K … … … … … 13%  573K 42s
4650K … … … … … 13%  575K 42s
4700K … … … … … 13%  573K 42s
4750K … … … … … 14%  694K 42s
4800K … … … … … 14%  958K 42s
4850K … … … … … 14%  924K 41s
4900K … … … … … 14%  574K 41s
4950K … … … … … 14%  573K 41s
5000K … … … … … 14%  571K 41s
5050K … … … … … 15% 1.31M 41s
5100K … … … … … 15%  952K 41s
5150K … … … … … 15%  569K 41s
5200K … … … … … 15%  580K 41s
5250K … … … … … 15% 1.30M 41s
5300K … … … … … 15%  929K 41s
5350K … … … … … 15%  562K 41s
5400K … … … … … 16%  583K 41s
5450K … … … … … 16%  576K 41s
5500K … … … … … 16% 1.31M 40s
5550K … … … … … 16%  596K 40s
5600K … … … … … 16%  865K 40s
5650K … … … … … 16%  573K 40s
5700K … … … … … 16%  576K 40s
5750K … … … … … 17%  690K 40s
5800K … … … … … 17%  967K 40s
5850K … … … … … 17%  924K 40s
5900K … … … … … 17%  574K 40s
5950K … … … … … 17%  706K 40s
6000K … … … … … 17%  949K 40s
6050K … … … … … 17%  928K 40s
6100K … … … … … 18%  570K 40s
6150K … … … … … 18%  575K 40s
6200K … … … … … 18%  574K 40s
6250K … … … … … 18% 1.30M 39s
6300K … … … … … 18%  930K 39s
6350K … … … … … 18%  575K 39s
6400K … … … … … 18%  574K 39s
6450K … … … … … 19%  575K 39s
6500K … … … … … 19% 1.25M 39s
6550K … … … … … 19%  578K 39s
```

```
6600K … … … … … 19%  935K 39s
6650K … … … … … 19%  574K 39s
6700K … … … … … 19%  573K 39s
6750K … … … … … 20%  573K 39s
6800K … … … … … 20% 17.4M 38s
6850K … … … … … 20%  573K 38s
6900K … … … … … 20%  576K 38s
6950K … … … … … 20%  574K 38s
7000K … … … … … 20% 1.29M 38s
7050K … … … … … 20%  934K 38s
7100K … … … … … 21%  573K 38s
7150K … … … … … 21%  574K 38s
7200K … … … … … 21%  573K 38s
7250K … … … … … 21% 1.29M 38s
7300K … … … … … 21%  588K 38s
7350K … … … … … 21%  888K 38s
7400K … … … … … 21%  575K 38s
7450K … … … … … 22%  573K 38s
7500K … … … … … 22%  687K 38s
7550K … … … … … 22%  960K 37s
7600K … … … … … 22%  932K 37s
7650K … … … … … 22%  583K 37s
7700K … … … … … 22%  686K 37s
7750K … … … … … 22%  973K 37s
7800K … … … … … 23%  922K 37s
7850K … … … … … 23%  573K 37s
7900K … … … … … 23%  575K 37s
7950K … … … … … 23%  572K 37s
8000K … … … … … 23% 1.29M 37s
8050K … … … … … 23%  933K 37s
8100K … … … … … 23%  574K 37s
8150K … … … … … 24%  572K 37s
8200K … … … … … 24%  573K 37s
8250K … … … … … 24% 1.26M 36s
8300K … … … … … 24%  585K 36s
8350K … … … … … 24%  921K 36s
8400K … … … … … 24%  569K 36s
8450K … … … … … 25%  576K 36s
8500K … … … … … 25% 1.36M 36s
8550K … … … … … 25%  896K 36s
8600K … … … … … 25%  585K 36s
8650K … … … … … 25%  575K 36s
8700K … … … … … 25%  570K 36s
8750K … … … … … 25% 1.32M 36s
8800K … … … … … 26%  927K 36s
8850K … … … … … 26%  571K 36s
8900K … … … … … 26%  574K 36s
8950K … … … … … 26%  575K 36s
```

```
 9000K … … … … … 26% 1.32M 35s
 9050K … … … … … 26%  581K 35s
 9100K … … … … … 26%  899K 35s
 9150K … … … … … 27%  576K 35s
 9200K … … … … … 27%  573K 35s
 9250K … … … … … 27%  678K 35s
 9300K … … … … … 27%  979K 35s
 9350K … … … … … 27%  948K 35s
 9400K … … … … … 27%  575K 35s
 9450K … … … … … 27%  677K 35s
 9500K … … … … … 28%  984K 35s
 9550K … … … … … 28%  923K 35s
 9600K … … … … … 28%  575K 34s
 9650K … … … … … 28%  574K 34s
 9700K … … … … … 28%  569K 34s
 9750K … … … … … 28% 1.31M 34s
 9800K … … … … … 28%  581K 34s
 9850K … … … … … 29%  914K 34s
 9900K … … … … … 29%  575K 34s
 9950K … … … … … 29%  572K 34s
10000K … … … … … 29% 1.27M 34s
10050K … … … … … 29%  583K 34s
10100K … … … … … 29%  918K 34s
10150K … … … … … 30%  576K 34s
10200K … … … … … 30% 1.27M 34s
10250K … … … … … 30%  589K 34s
10300K … … … … … 30%  921K 33s
10350K … … … … … 30%  574K 33s
10400K … … … … … 30%  573K 33s
10450K … … … … … 30%  680K 33s
10500K … … … … … 31%  970K 33s
10550K … … … … … 31%  888K 33s
10600K … … … … … 31%  595K 33s
10650K … … … … … 31%  573K 33s
10700K … … … … … 31%  573K 33s
10750K … … … … … 31% 1.25M 33s
10800K … … … … … 31%  587K 33s
10850K … … … … … 32%  917K 33s
10900K … … … … … 32%  575K 33s
10950K … … … … … 32%  572K 33s
11000K … … … … … 32%  717K 33s
11050K … … … … … 32% 2.38M 32s
11100K … … … … … 32%  573K 32s
11150K … … … … … 32%  564K 32s
11200K … … … … … 33%  735K 32s
11250K … … … … … 33%  909K 32s
11300K … … … … … 33%  929K 32s
11350K … … … … … 33%  574K 32s
```

```
11400K … … … … … 33%  575K 32s
11450K … … … … … 33%  572K 32s
11500K … … … … … 33% 1.32M 32s
11550K … … … … … 34%  567K 32s
11600K … … … … … 34%  941K 32s
11650K … … … … … 34%  572K 32s
11700K … … … … … 34%  576K 32s
11750K … … … … … 34% 1.25M 31s
11800K … … … … … 34%  586K 31s
11850K … … … … … 35%  921K 31s
11900K … … … … … 35%  583K 31s
11950K … … … … … 35% 1.25M 31s
12000K … … … … … 35%  583K 31s
12050K … … … … … 35%  922K 31s
12100K … … … … … 35%  576K 31s
12150K … … … … … 35%  572K 31s
12200K … … … … … 36%  670K 31s
12250K … … … … … 36% 1005K 31s
12300K … … … … … 36%  919K 31s
12350K … … … … … 36%  573K 31s
12400K … … … … … 36%  574K 31s
12450K … … … … … 36%  574K 30s
12500K … … … … … 36% 1.26M 30s
12550K … … … … … 37%  580K 30s
12600K … … … … … 37%  925K 30s
12650K … … … … … 37%  577K 30s
12700K … … … … … 37%  572K 30s
12750K … … … … … 37% 1.38M 30s
12800K … … … … … 37%  913K 30s
12850K … … … … … 37%  572K 30s
12900K … … … … … 38%  576K 30s
12950K … … … … … 38%  710K 30s
13000K … … … … … 38%  918K 30s
13050K … … … … … 38%  927K 30s
13100K … … … … … 38%  576K 29s
13150K … … … … … 38%  573K 29s
13200K … … … … … 38%  573K 29s
13250K … … … … … 39% 1.31M 29s
13300K … … … … … 39%  571K 29s
13350K … … … … … 39%  930K 29s
13400K … … … … … 39%  573K 29s
13450K … … … … … 39%  572K 29s
13500K … … … … … 39%  712K 29s
13550K … … … … … 40%  938K 29s
13600K … … … … … 40%  936K 29s
13650K … … … … … 40%  575K 29s
13700K … … … … … 40% 1.27M 29s
13750K … … … … … 40%  585K 29s
```

```
13800K … … … … … 40%  914K 28s
13850K … … … … … 40%  574K 28s
13900K … … … … … 41%  574K 28s
13950K … … … … … 41%  672K 28s
14000K … … … … … 41%  999K 28s
14050K … … … … … 41%  925K 28s
14100K … … … … … 41%  573K 28s
14150K … … … … … 41%  573K 28s
14200K … … … … … 41%  571K 28s
14250K … … … … … 42% 1.28M 28s
14300K … … … … … 42%  584K 28s
14350K … … … … … 42%  911K 28s
14400K … … … … … 42%  575K 28s
14450K … … … … … 42% 1.30M 28s
14500K … … … … … 42%  588K 27s
14550K … … … … … 42%  920K 27s
14600K … … … … … 43%  573K 27s
14650K … … … … … 43%  574K 27s
14700K … … … … … 43%  698K 27s
14750K … … … … … 43%  943K 27s
14800K … … … … … 43%  927K 27s
14850K … … … … … 43%  573K 27s
14900K … … … … … 43%  574K 27s
14950K … … … … … 44%  571K 27s
15000K … … … … … 44% 1.33M 27s
15050K … … … … … 44%  572K 27s
15100K … … … … … 44%  913K 27s
15150K … … … … … 44%  579K 27s
15200K … … … … … 44%  572K 27s
15250K … … … … … 45%  699K 26s
15300K … … … … … 45% 2.60M 26s
15350K … … … … … 45%  576K 26s
15400K … … … … … 45%  571K 26s
15450K … … … … … 45% 1.26M 26s
15500K … … … … … 45%  588K 26s
15550K … … … … … 45%  925K 26s
15600K … … … … … 46%  569K 26s
15650K … … … … … 46%  576K 26s
15700K … … … … … 46%  667K 26s
15750K … … … … … 46% 1012K 26s
15800K … … … … … 46%  925K 26s
15850K … … … … … 46%  574K 26s
15900K … … … … … 46%  572K 26s
15950K … … … … … 47%  565K 25s
16000K … … … … … 47% 1.28M 25s
16050K … … … … … 47%  588K 25s
16100K … … … … … 47%  922K 25s
16150K … … … … … 47%  575K 25s
```

```
16200K … … … … … 47% 1.28M 25s
16250K … … … … … 47%  587K 25s
16300K … … … … … 48%  921K 25s
16350K … … … … … 48%  574K 25s
16400K … … … … … 48%  572K 25s
16450K … … … … … 48%  707K 25s
16500K … … … … … 48%  935K 25s
16550K … … … … … 48%  907K 25s
16600K … … … … … 48%  578K 25s
16650K … … … … … 49%  573K 24s
16700K … … … … … 49%  574K 24s
16750K … … … … … 49% 1.31M 24s
16800K … … … … … 49%  572K 24s
16850K … … … … … 49%  921K 24s
16900K … … … … … 49%  575K 24s
16950K … … … … … 50%  576K 24s
17000K … … … … … 50% 1.35M 24s
17050K … … … … … 50%  923K 24s
17100K … … … … … 50%  573K 24s
17150K … … … … … 50%  573K 24s
17200K … … … … … 50%  700K 24s
17250K … … … … … 50%  946K 24s
17300K … … … … … 51%  919K 24s
17350K … … … … … 51%  574K 23s
17400K … … … … … 51%  575K 23s
17450K … … … … … 51%  661K 23s
17500K … … … … … 51% 1.01M 23s
17550K … … … … … 51%  925K 23s
17600K … … … … … 51%  572K 23s
17650K … … … … … 52%  571K 23s
17700K … … … … … 52%  578K 23s
17750K … … … … … 52% 1.25M 23s
17800K … … … … … 52%  585K 23s
17850K … … … … … 52%  947K 23s
17900K … … … … … 52%  571K 23s
17950K … … … … … 52% 1.26M 23s
18000K … … … … … 53%  584K 23s
18050K … … … … … 53%  927K 22s
18100K … … … … … 53%  573K 22s
18150K … … … … … 53%  575K 22s
18200K … … … … … 53%  698K 22s
18250K … … … … … 53%  942K 22s
18300K … … … … … 53%  875K 22s
18350K … … … … … 54%  585K 22s
18400K … … … … … 54%  573K 22s
18450K … … … … … 54%  585K 22s
18500K … … … … … 54% 1.30M 22s
18550K … … … … … 54%  573K 22s
```

```
18600K … … … … … 54%  876K 22s
18650K … … … … … 55%  584K 22s
18700K … … … … … 55% 1.41M 22s
18750K … … … … … 55%  573K 21s
18800K … … … … … 55%  902K 21s
18850K … … … … … 55%  574K 21s
18900K … … … … … 55%  572K 21s
18950K … … … … … 55%  712K 21s
19000K … … … … … 56%  946K 21s
19050K … … … … … 56%  893K 21s
19100K … … … … … 56%  575K 21s
19150K … … … … … 56%  578K 21s
19200K … … … … … 56%  665K 21s
19250K … … … … … 56% 1.01M 21s
19300K … … … … … 56%  892K 21s
19350K … … … … … 57%  577K 21s
19400K … … … … … 57%  574K 21s
19450K … … … … … 57%  581K 21s
19500K … … … … … 57% 1.22M 20s
19550K … … … … … 57%  977K 20s
19600K … … … … … 57%  573K 20s
19650K … … … … … 57%  578K 20s
19700K … … … … … 58% 1.23M 20s
19750K … … … … … 58%  593K 20s
19800K … … … … … 58%  899K 20s
19850K … … … … … 58%  574K 20s
19900K … … … … … 58%  574K 20s
19950K … … … … … 58%  664K 20s
20000K … … … … … 58% 1.02M 20s
20050K … … … … … 59%  900K 20s
20100K … … … … … 59%  576K 20s
20150K … … … … … 59%  574K 20s
20200K … … … … … 59%  578K 19s
20250K … … … … … 59% 1.31M 19s
20300K … … … … … 59%  574K 19s
20350K … … … … … 60%  904K 19s
20400K … … … … … 60%  587K 19s
20450K … … … … … 60% 1.33M 19s
20500K … … … … … 60%  572K 19s
20550K … … … … … 60%  909K 19s
20600K … … … … … 60%  573K 19s
20650K … … … … … 60%  576K 19s
20700K … … … … … 61%  694K 19s
20750K … … … … … 61%  963K 19s
20800K … … … … … 61%  910K 19s
20850K … … … … … 61%  574K 19s
20900K … … … … … 61%  575K 18s
20950K … … … … … 61%  656K 18s
```

```
21000K … … … … … 61% 1.03M 18s
21050K … … … … … 62%  903K 18s
21100K … … … … … 62%  566K 18s
21150K … … … … … 62%  582K 18s
21200K … … … … … 62%  579K 18s
21250K … … … … … 62% 13.4M 18s
21300K … … … … … 62%  572K 18s
21350K … … … … … 62%  571K 18s
21400K … … … … … 63%  581K 18s
21450K … … … … … 63% 1.21M 18s
21500K … … … … … 63%  596K 18s
21550K … … … … … 63%  852K 18s
21600K … … … … … 63%  597K 17s
21650K … … … … … 63%  577K 17s
21700K … … … … … 63%  654K 17s
21750K … … … … … 64% 1.03M 17s
21800K … … … … … 64%  902K 17s
21850K … … … … … 64%  575K 17s
21900K … … … … … 64%  579K 17s
21950K … … … … … 64%  577K 17s
22000K … … … … … 64% 1.28M 17s
22050K … … … … … 65%  575K 17s
22100K … … … … … 65%  934K 17s
22150K … … … … … 65%  579K 17s
22200K … … … … … 65% 1.30M 17s
22250K … … … … … 65%  574K 17s
22300K … … … … … 65%  912K 16s
22350K … … … … … 65%  569K 16s
22400K … … … … … 66%  578K 16s
22450K … … … … … 66%  702K 16s
22500K … … … … … 66%  956K 16s
22550K … … … … … 66%  910K 16s
22600K … … … … … 66%  569K 16s
22650K … … … … … 66%  581K 16s
22700K … … … … … 66%  573K 16s
22750K … … … … … 67% 1.30M 16s
22800K … … … … … 67%  929K 16s
22850K … … … … … 67%  573K 16s
22900K … … … … … 67%  574K 16s
22950K … … … … … 67% 1.33M 16s
23000K … … … … … 67%  929K 15s
23050K … … … … … 67%  573K 15s
23100K … … … … … 68%  574K 15s
23150K … … … … … 68%  575K 15s
23200K … … … … … 68% 1.24M 15s
23250K … … … … … 68%  584K 15s
23300K … … … … … 68%  929K 15s
23350K … … … … … 68%  568K 15s
```

```
23400K … … … … … 68%  580K 15s
23450K … … … … … 69%  654K 15s
23500K … … … … … 69% 1.01M 15s
23550K … … … … … 69%  920K 15s
23600K … … … … … 69%  569K 15s
23650K … … … … … 69%  582K 15s
23700K … … … … … 69%  574K 14s
23750K … … … … … 70% 1.29M 14s
23800K … … … … … 70%  934K 14s
23850K … … … … … 70%  582K 14s
23900K … … … … … 70%  572K 14s
23950K … … … … … 70% 1.27M 14s
24000K … … … … … 70%  574K 14s
24050K … … … … … 70%  937K 14s
24100K … … … … … 71%  569K 14s
24150K … … … … … 71%  576K 14s
24200K … … … … … 71%  690K 14s
24250K … … … … … 71%  959K 14s
24300K … … … … … 71%  921K 14s
24350K … … … … … 71%  574K 14s
24400K … … … … … 71%  580K 13s
24450K … … … … … 72%  570K 13s
24500K … … … … … 72% 1.29M 13s
24550K … … … … … 72%  568K 13s
24600K … … … … … 72%  939K 13s
24650K … … … … … 72%  589K 13s
24700K … … … … … 72% 1.27M 13s
24750K … … … … … 72%  921K 13s
24800K … … … … … 73%  576K 13s
24850K … … … … … 73%  570K 13s
24900K … … … … … 73%  580K 13s
24950K … … … … … 73% 1.24M 13s
25000K … … … … … 73%  582K 13s
25050K … … … … … 73%  915K 13s
25100K … … … … … 73%  578K 12s
25150K … … … … … 74%  581K 12s
25200K … … … … … 74%  660K 12s
25250K … … … … … 74% 1011K 12s
25300K … … … … … 74%  898K 12s
25350K … … … … … 74%  578K 12s
25400K … … … … … 74%  578K 12s
25450K … … … … … 75%  580K 12s
25500K … … … … … 75% 11.2M 12s
25550K … … … … … 75%  579K 12s
25600K … … … … … 75%  579K 12s
25650K … … … … … 75%  574K 12s
25700K … … … … … 75% 1.27M 12s
25750K … … … … … 75%  569K 12s
```

```
25800K … … … … … 76%  937K 12s
25850K … … … … … 76%  571K 11s
25900K … … … … … 76%  579K 11s
25950K … … … … … 76%  460K 11s
26000K … … … … … 76% 2.57M 11s
26050K … … … … … 76%  970K 11s
26100K … … … … … 76%  566K 11s
26150K … … … … … 77%  585K 11s
26200K … … … … … 77%  577K 11s
26250K … … … … … 77% 1.27M 11s
26300K … … … … … 77%  571K 11s
26350K … … … … … 77%  972K 11s
26400K … … … … … 77%  569K 11s
26450K … … … … … 77% 1.29M 11s
26500K … … … … … 78%  919K 11s
26550K … … … … … 78%  575K 10s
26600K … … … … … 78%  573K 10s
26650K … … … … … 78%  572K 10s
26700K … … … … … 78% 1.27M 10s
26750K … … … … … 78%  582K 10s
26800K … … … … … 78%  931K 10s
26850K … … … … … 79%  573K 10s
26900K … … … … … 79%  583K 10s
26950K … … … … … 79%  646K 10s
27000K … … … … … 79% 1.01M 10s
27050K … … … … … 79%  918K 10s
27100K … … … … … 79%  578K 10s
27150K … … … … … 80%  572K 10s
27200K … … … … … 80% 1.34M 10s
27250K … … … … … 80%  930K 9s
27300K … … … … … 80%  575K 9s
27350K … … … … … 80%  578K 9s
27400K … … … … … 80%  574K 9s
27450K … … … … … 80% 1.27M 9s
27500K … … … … … 81%  574K 9s
27550K … … … … … 81%  931K 9s
27600K … … … … … 81%  570K 9s
27650K … … … … … 81%  583K 9s
27700K … … … … … 81%  680K 9s
27750K … … … … … 81%  964K 9s
27800K … … … … … 81%  935K 9s
27850K … … … … … 82%  572K 9s
27900K … … … … … 82%  580K 9s
27950K … … … … … 82%  571K 8s
28000K … … … … … 82% 1.27M 8s
28050K … … … … … 82%  954K 8s
28100K … … … … … 82%  571K 8s
28150K … … … … … 82%  586K 8s
```

```
28200K … … … … … 83% 1.26M 8s
28250K … … … … … 83%  936K 8s
28300K … … … … … 83%  574K 8s
28350K … … … … … 83%  571K 8s
28400K … … … … … 83%  581K 8s
28450K … … … … … 83% 1.20M 8s
28500K … … … … … 83%  582K 8s
28550K … … … … … 84%  948K 8s
28600K … … … … … 84%  568K 8s
28650K … … … … … 84%  584K 7s
28700K … … … … … 84%  637K 7s
28750K … … … … … 84% 1.01M 7s
28800K … … … … … 84%  943K 7s
28850K … … … … … 85%  567K 7s
28900K … … … … … 85%  668K 7s
28950K … … … … … 85% 1.00M 7s
29000K … … … … … 85%  924K 7s
29050K … … … … … 85%  521K 7s
29100K … … … … … 85%  650K 7s
29150K … … … … … 85%  579K 7s
29200K … … … … … 86% 1.21M 7s
29250K … … … … … 86%  575K 7s
29300K … … … … … 86%  953K 7s
29350K … … … … … 86%  576K 6s
29400K … … … … … 86%  579K 6s
29450K … … … … … 86%  677K 6s
29500K … … … … … 86%  946K 6s
29550K … … … … … 87%  951K 6s
29600K … … … … … 87%  570K 6s
29650K … … … … … 87%  582K 6s
29700K … … … … … 87%  571K 6s
29750K … … … … … 87% 13.4M 6s
29800K … … … … … 87%  575K 6s
29850K … … … … … 87%  578K 6s
29900K … … … … … 88%  573K 6s
29950K … … … … … 88% 1.23M 6s
30000K … … … … … 88%  573K 6s
30050K … … … … … 88%  958K 6s
30100K … … … … … 88%  574K 5s
30150K … … … … … 88%  579K 5s
30200K … … … … … 88% 1.22M 5s
30250K … … … … … 89%  574K 5s
30300K … … … … … 89%  953K 5s
30350K … … … … … 89%  576K 5s
30400K … … … … … 89%  579K 5s
30450K … … … … … 89%  637K 5s
30500K … … … … … 89% 1017K 5s
30550K … … … … … 90%  962K 5s
```

```
30600K … … … … … 90%  590K 5s
30650K … … … … … 90%  634K 5s
30700K … … … … … 90% 1.00M 5s
30750K … … … … … 90%  955K 5s
30800K … … … … … 90%  576K 4s
30850K … … … … … 90%  571K 4s
30900K … … … … … 91%  580K 4s
30950K … … … … … 91% 1.22M 4s
31000K … … … … … 91%  572K 4s
31050K … … … … … 91%  966K 4s
31100K … … … … … 91%  573K 4s
31150K … … … … … 91%  579K 4s
31200K … … … … … 91%  679K 4s
31250K … … … … … 92%  927K 4s
31300K … … … … … 92%  965K 4s
31350K … … … … … 92%  575K 4s
31400K … … … … … 92%  577K 4s
31450K … … … … … 92% 1.25M 4s
31500K … … … … … 92%  973K 3s
31550K … … … … … 92%  576K 3s
31600K … … … … … 93%  574K 3s
31650K … … … … … 93%  572K 3s
31700K … … … … … 93% 1.22M 3s
31750K … … … … … 93%  576K 3s
31800K … … … … … 93%  971K 3s
31850K … … … … … 93%  573K 3s
31900K … … … … … 93%  576K 3s
31950K … … … … … 94% 1.22M 3s
32000K … … … … … 94%  573K 3s
32050K … … … … … 94%  966K 3s
32100K … … … … … 94%  574K 3s
32150K … … … … … 94%  573K 3s
32200K … … … … … 94%  646K 2s
32250K … … … … … 95% 1013K 2s
32300K … … … … … 95%  985K 2s
32350K … … … … … 95%  577K 2s
32400K … … … … … 95%  642K 2s
32450K … … … … … 95% 1016K 2s
32500K … … … … … 95%  964K 2s
32550K … … … … … 95%  573K 2s
32600K … … … … … 96%  576K 2s
32650K … … … … … 96%  576K 2s
32700K … … … … … 96% 1.22M 2s
32750K … … … … … 96%  570K 2s
32800K … … … … … 96%  972K 2s
32850K … … … … … 96%  573K 2s
32900K … … … … … 96%  578K 1s
32950K … … … … … 97%  678K 1s
```

```
 33000K .......... .......... .......... .......... .......... 97%  939K 1s
 33050K .......... .......... .......... .......... .......... 97%  965K 1s
 33100K .......... .......... .......... .......... .......... 97%  572K 1s
 33150K .......... .......... .......... .......... .......... 97%  653K 1s
 33200K .......... .......... .......... .......... .......... 97% 1007K 1s
 33250K .......... .......... .......... .......... .......... 97%  980K 1s
 33300K .......... .......... .......... .......... .......... 98%  563K 1s
 33350K .......... .......... .......... .......... .......... 98%  589K 1s
 33400K .......... .......... .......... .......... .......... 98%  571K 1s
 33450K .......... .......... .......... .......... .......... 98% 1.15M 1s
 33500K .......... .......... .......... .......... .......... 98%  589K 1s
 33550K .......... .......... .......... .......... .......... 98%  970K 1s
 33600K .......... .......... .......... .......... .......... 98%  575K 0s
 33650K .......... .......... .......... .......... .......... 99%  575K 0s
 33700K .......... .......... .......... .......... .......... 99%  677K 0s
 33750K .......... .......... .......... .......... .......... 99%  935K 0s
 33800K .......... .......... .......... .......... .......... 99%  970K 0s
 33850K .......... .......... .......... .......... .......... 99%  573K 0s
 33900K .......... .......... .......... .......... .......... 99%  576K 0s
 33950K .......... .......... .......... .......... ..........    100% 1.18M=48s

2023-03-09 15:41:59 (708 KB/s) - 'MNIST.tar.gz' saved [34812527/34812527]

x MNIST/
x MNIST/raw/
x MNIST/raw/train-labels-idx1-ubyte.gz
x MNIST/raw/t10k-images-idx3-ubyte
x MNIST/raw/train-images-idx3-ubyte
x MNIST/raw/t10k-labels-idx1-ubyte.gz
x MNIST/raw/train-images-idx3-ubyte.gz
x MNIST/raw/t10k-images-idx3-ubyte.gz
x MNIST/raw/train-labels-idx1-ubyte
x MNIST/raw/t10k-labels-idx1-ubyte
x MNIST/processed/
x MNIST/processed/test.pt
x MNIST/processed/training.pt
```

1.2 Exercise 1: Self-Attention without Positional Encoding

In this section, will implement a very simple model based on self-attention without positional encoding. The model you will implement will consider the input image as a sequence of 28 rows. You may use PyTorch's `nn.MultiheadAttention` for this part. Implement a model with the following architecture:

- **Input**: Input image of shape `(batch_size, sequence_length, input_size)`, where sequence_length = image_height and input_size = image_width.
- **Linear 1**: Linear layer which converts input of shape `(sequence_length*batch_size, input_size)` to input of shape `(sequence_length*batch_size, embed_dim)`, where

`embed_dim` is the embedding dimension.

- **Attention 1**: `nn.MultiheadAttention` layer with 8 heads which takes an input of shape `(sequence_length, batch_size, embed_dim)` and outputs a tensor of shape `(sequence_length, batch_size, embed_dim)`.
- **ReLU**: ReLU activation layer.
- **Linear 2**: Linear layer which converts input of shape `(sequence_length*batch_size, embed_dim)` to input of shape `(sequence_length*batch_size, embed_dim)`.
- **ReLU**: ReLU activation layer.
- **Attention 2**: `nn.MultiheadAttention` layer with 8 heads which takes an input of shape `(sequence_length, batch_size, embed_dim)` and outputs a tensor of shape `(sequence_length, batch_size, embed_dim)`.
- **ReLU**: ReLU activation layer.
- **AvgPool**: Average along the sequence dimension from `(batch_size, sequence_length, embed_dim)` to `(batch_size, embed_dim)`
- **Linear 3**: Linear layer which takes an input of shape `(batch_size, embed_dim)` and outputs the class logits of shape `(batch_size, 10)`.

NOTE: Be cautious of correctly permuting and reshaping the input between layers. E.g. if `x` is of shape `(batch_size, sequence_length, input_size)`, note that `x.reshape(batch_size*sequence_length, -1) != x.permute(1,0,2).reshape(batch_size*sequence_length, -1)`. In this example, `x.reshape(batch_size*sequence_length, -1)` has `[batch0_seq0, batch0_seq1, ..., batch1_seq0, batch1_seq1, ...]` format, while `x.permute(1,0,2).reshape(batch_size*sequence_length, -1)` has `[batch0_seq0, batch1_seq0, ..., batch0_seq1, batch1_seq1, ...]` format.

[ ]:
```python
# Self-attention without positional encoding
torch.manual_seed(691)

# Define your model here
class myModel(nn.Module):
    def __init__(self, input_size, embed_dim, seq_length,
                 num_classes=10, num_heads=8):
        super(myModel, self).__init__()
        self.layer1 = nn.Linear(input_size, embed_dim)
        self.attention1 = nn.MultiheadAttention(embed_dim, num_heads)
        self.layer2 = nn.Linear(embed_dim, embed_dim)
        self.attention2 = nn.MultiheadAttention(embed_dim, num_heads)
        # From 64*28*64 to 64*64 using average pooling
        self.avgpool = nn.AvgPool1d(kernel_size=seq_length)
        self.fc = nn.Linear(embed_dim, num_classes)
        self.activation = nn.ReLU()
```

```python
    def forward(self,x):
        # TODO: Implement myModel forward pass
        batch_size, sequence_length, input_size = x.shape
        x = x.reshape(batch_size*sequence_length, input_size)
        x = self.layer1(x)
        x = self.activation(x)
        x = x.reshape(batch_size, sequence_length, -1)
        x = x.permute(1, 0, 2)
        x, _ = self.attention1(x, x, x)
        x = x.permute(1, 0, 2)
        x = x.reshape(batch_size*sequence_length, -1)
        x = self.layer2(x)
        x = self.activation(x)
        x = x.reshape(batch_size, sequence_length, -1)
        x = x.permute(1, 0, 2)
        x, _ = self.attention2(x, x, x)
        x = x.permute(1, 0, 2)
        x = x.reshape(batch_size,sequence_length, -1)
        # From 64*28*64 to 64*64 using average pooling
        x = x.permute(0, 2, 1)
        x = self.avgpool(x).squeeze()
        x = self.fc(x)
        return x
```

Train and evaluate your model by running the cell below. Expect to see `60-80%` test accuracy.

[ ]:
```python
# Same training code

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms


# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyper-parameters
sequence_length = 28
input_size = 28
hidden_size = 64
num_layers = 2
num_classes = 10
num_epochs = 8
learning_rate = 0.005

# Initialize model
```

```python
model = myModel(input_size=input_size, embed_dim=hidden_size,␣
 ↪seq_length=sequence_length)
model = model.to(device)
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()

        optimizer.step()

        if (i+1) % 10 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))


# Test the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Test Accuracy of the model on the 10000 test images: {} %'.
 ↪format(100 * correct / total))
```

```
Epoch [1/8], Step [10/157], Loss: 2.2581
Epoch [1/8], Step [20/157], Loss: 2.1685
Epoch [1/8], Step [30/157], Loss: 1.9935
```

```
Epoch [1/8], Step [40/157], Loss: 1.8937
Epoch [1/8], Step [50/157], Loss: 1.9072
Epoch [1/8], Step [60/157], Loss: 1.5847
Epoch [1/8], Step [70/157], Loss: 1.7249
Epoch [1/8], Step [80/157], Loss: 1.7521
Epoch [1/8], Step [90/157], Loss: 1.6420
Epoch [1/8], Step [100/157], Loss: 1.6267
Epoch [1/8], Step [110/157], Loss: 1.5332
Epoch [1/8], Step [120/157], Loss: 1.5796
Epoch [1/8], Step [130/157], Loss: 1.4901
Epoch [1/8], Step [140/157], Loss: 1.4746
Epoch [1/8], Step [150/157], Loss: 1.4116
Epoch [2/8], Step [10/157], Loss: 1.2400
Epoch [2/8], Step [20/157], Loss: 1.3711
Epoch [2/8], Step [30/157], Loss: 1.2489
Epoch [2/8], Step [40/157], Loss: 1.1527
Epoch [2/8], Step [50/157], Loss: 1.0967
Epoch [2/8], Step [60/157], Loss: 0.9711
Epoch [2/8], Step [70/157], Loss: 0.9957
Epoch [2/8], Step [80/157], Loss: 0.8141
Epoch [2/8], Step [90/157], Loss: 0.8850
Epoch [2/8], Step [100/157], Loss: 0.8135
Epoch [2/8], Step [110/157], Loss: 0.6558
Epoch [2/8], Step [120/157], Loss: 1.0183
Epoch [2/8], Step [130/157], Loss: 0.9859
Epoch [2/8], Step [140/157], Loss: 1.1008
Epoch [2/8], Step [150/157], Loss: 1.0552
Epoch [3/8], Step [10/157], Loss: 0.8491
Epoch [3/8], Step [20/157], Loss: 0.9709
Epoch [3/8], Step [30/157], Loss: 0.8101
Epoch [3/8], Step [40/157], Loss: 0.6358
Epoch [3/8], Step [50/157], Loss: 0.8498
Epoch [3/8], Step [60/157], Loss: 0.8543
Epoch [3/8], Step [70/157], Loss: 0.8947
Epoch [3/8], Step [80/157], Loss: 1.0854
Epoch [3/8], Step [90/157], Loss: 0.7056
Epoch [3/8], Step [100/157], Loss: 0.5561
Epoch [3/8], Step [110/157], Loss: 0.7541
Epoch [3/8], Step [120/157], Loss: 0.9077
Epoch [3/8], Step [130/157], Loss: 0.8249
Epoch [3/8], Step [140/157], Loss: 0.7715
Epoch [3/8], Step [150/157], Loss: 0.7841
Epoch [4/8], Step [10/157], Loss: 0.9479
Epoch [4/8], Step [20/157], Loss: 0.9176
Epoch [4/8], Step [30/157], Loss: 0.6106
Epoch [4/8], Step [40/157], Loss: 0.5420
Epoch [4/8], Step [50/157], Loss: 0.8178
Epoch [4/8], Step [60/157], Loss: 0.9015
```

```
Epoch [4/8], Step [70/157], Loss: 0.6783
Epoch [4/8], Step [80/157], Loss: 0.6113
Epoch [4/8], Step [90/157], Loss: 0.6901
Epoch [4/8], Step [100/157], Loss: 0.8723
Epoch [4/8], Step [110/157], Loss: 0.9989
Epoch [4/8], Step [120/157], Loss: 0.6024
Epoch [4/8], Step [130/157], Loss: 0.5923
Epoch [4/8], Step [140/157], Loss: 0.6748
Epoch [4/8], Step [150/157], Loss: 0.6014
Epoch [5/8], Step [10/157], Loss: 0.6849
Epoch [5/8], Step [20/157], Loss: 0.5970
Epoch [5/8], Step [30/157], Loss: 0.7019
Epoch [5/8], Step [40/157], Loss: 0.5989
Epoch [5/8], Step [50/157], Loss: 0.6658
Epoch [5/8], Step [60/157], Loss: 0.7669
Epoch [5/8], Step [70/157], Loss: 0.5594
Epoch [5/8], Step [80/157], Loss: 0.4520
Epoch [5/8], Step [90/157], Loss: 0.4293
Epoch [5/8], Step [100/157], Loss: 0.5613
Epoch [5/8], Step [110/157], Loss: 0.6237
Epoch [5/8], Step [120/157], Loss: 0.6710
Epoch [5/8], Step [130/157], Loss: 0.6414
Epoch [5/8], Step [140/157], Loss: 0.7120
Epoch [5/8], Step [150/157], Loss: 0.5976
Epoch [6/8], Step [10/157], Loss: 0.4132
Epoch [6/8], Step [20/157], Loss: 0.5664
Epoch [6/8], Step [30/157], Loss: 0.3627
Epoch [6/8], Step [40/157], Loss: 0.4754
Epoch [6/8], Step [50/157], Loss: 0.7253
Epoch [6/8], Step [60/157], Loss: 0.7572
Epoch [6/8], Step [70/157], Loss: 0.4816
Epoch [6/8], Step [80/157], Loss: 0.5277
Epoch [6/8], Step [90/157], Loss: 0.5299
Epoch [6/8], Step [100/157], Loss: 0.5431
Epoch [6/8], Step [110/157], Loss: 0.5689
Epoch [6/8], Step [120/157], Loss: 0.4726
Epoch [6/8], Step [130/157], Loss: 0.5370
Epoch [6/8], Step [140/157], Loss: 0.5600
Epoch [6/8], Step [150/157], Loss: 0.4915
Epoch [7/8], Step [10/157], Loss: 0.4666
Epoch [7/8], Step [20/157], Loss: 0.5318
Epoch [7/8], Step [30/157], Loss: 0.4142
Epoch [7/8], Step [40/157], Loss: 0.4947
Epoch [7/8], Step [50/157], Loss: 0.2787
Epoch [7/8], Step [60/157], Loss: 0.7696
Epoch [7/8], Step [70/157], Loss: 0.2714
Epoch [7/8], Step [80/157], Loss: 0.8138
Epoch [7/8], Step [90/157], Loss: 0.4571
```

```
Epoch [7/8], Step [100/157], Loss: 0.7002
Epoch [7/8], Step [110/157], Loss: 0.5610
Epoch [7/8], Step [120/157], Loss: 0.3775
Epoch [7/8], Step [130/157], Loss: 0.6092
Epoch [7/8], Step [140/157], Loss: 0.3784
Epoch [7/8], Step [150/157], Loss: 0.5044
Epoch [8/8], Step [10/157], Loss: 0.4760
Epoch [8/8], Step [20/157], Loss: 0.5610
Epoch [8/8], Step [30/157], Loss: 0.3913
Epoch [8/8], Step [40/157], Loss: 0.3071
Epoch [8/8], Step [50/157], Loss: 0.5538
Epoch [8/8], Step [60/157], Loss: 0.4897
Epoch [8/8], Step [70/157], Loss: 0.4687
Epoch [8/8], Step [80/157], Loss: 0.5325
Epoch [8/8], Step [90/157], Loss: 0.5411
Epoch [8/8], Step [100/157], Loss: 0.3775
Epoch [8/8], Step [110/157], Loss: 0.3030
Epoch [8/8], Step [120/157], Loss: 0.5804
Epoch [8/8], Step [130/157], Loss: 0.6735
Epoch [8/8], Step [140/157], Loss: 0.4339
Epoch [8/8], Step [150/157], Loss: 0.3462
Test Accuracy of the model on the 10000 test images: 82.88 %
```

1.3 Exercise 2: Self-Attention with Positional Encoding

Implement a similar model to exercise 1, except this time your embedded input should be added with the positional encoding. For the purpose of this lab, we will use a learned positional encoding, which will be a trainable embedding. Your positional encodings will be added to the initial transformation of the input.

- **Input**: Input image of shape `(batch_size, sequence_length, input_size)`, where sequence_length = image_height and input_size = image_width.
- **Linear 1**: Linear layer which converts input of shape `(batch_size*sequence_length, input_size)` to input of shape `(batch_size*sequence_length, embed_dim)`, where `embed_dim` is the embedding dimension.
- **Add Positional Encoding**: Add a learnable positional encoding of shape `(sequence_length, batch_size, embed_dim)` to input of shape `(sequence_length, batch_size, embed_dim)`, where `pos_embed` is the positional embedding size. The output will be of shape `(sequence_length, batch_size, embed_dim)`.
- **Attention 1**: `nn.MultiheadAttention` layer with 8 heads which takes an input of shape `(sequence_length, batch_size, embed_dim)` and outputs a tensor of shape `(sequence_length, batch_size, embed_dim)`.
- **ReLU**: ReLU activation layer.
- **Linear 2**: Linear layer which converts input of shape `(sequence_length*batch_size, features_dim)` to input of shape `(sequence_length*batch_size, features_dim)`.

- **ReLU**: ReLU activation layer.
- **Attention 2**: `nn.MultiheadAttention` layer with 8 heads which takes an input of shape `(sequence_length, batch_size, features_dim)` and outputs a tensor of shape `(sequence_length, batch_size, features_dim)`.
- **ReLU**: ReLU activation layer.
- **AvgPool**: Average along the sequence dimension from `(batch_size, sequence_length, features_dim)` to `(batch_size, features_dim)`
- **Linear 3**: Linear layer which takes an input of shape `(batch_size, sequence_length*features_dim)` and outputs the class logits of shape `(batch_size, 10)`.

[ ]:
```
# Self-attention with positional encoding
torch.manual_seed(691)

# Define your model here
class myModel(nn.Module):
    def __init__(self, input_size, embed_dim, seq_length,
                 num_classes=10, num_heads=8):
        super(myModel, self).__init__()

        self.seq_length = seq_length
        self.embed_dim = embed_dim

        self.layer1 = nn.Linear(input_size, embed_dim)
        self.positional_encoding = nn.Parameter(torch.rand(self.seq_length,␣
↪self.embed_dim))
        self.attention1 = nn.MultiheadAttention(embed_dim, num_heads)
        self.layer2 = nn.Linear(embed_dim, embed_dim)
        self.attention2 = nn.MultiheadAttention(embed_dim, num_heads)
        self.avgpool = nn.AvgPool1d(kernel_size=seq_length)
        self.fc = nn.Linear(seq_length*embed_dim, num_classes)
        self.activation = nn.ReLU()


    def forward(self,x):
        # TODO: Implement myModel forward pass
        batch_size, sequence_length, input_size = x.shape
        x = x.reshape(batch_size*sequence_length, input_size)
        x = self.layer1(x)
        x = x.reshape(batch_size, sequence_length, -1)
        x = x.permute(1, 0, 2)
        # Add positional encoding, positional encoding is (seq_length,␣
↪embed_dim), reshape to (seq_length, batch_size, embed_dim)
        x = x + self.positional_encoding.reshape(sequence_length, 1, -1)
        x, _ = self.attention1(x, x, x)
```

```
        x = self.activation(x)
        x = x.reshape(sequence_length*batch_size, -1)
        x = self.layer2(x)
        x = self.activation(x)
        x = x.reshape(sequence_length, batch_size, -1)
        x , _ = self.attention2(x, x, x)
        x = self.activation(x)
        #print (x.shape)
        x = x.permute(1,2,0)
        #print (x.shape)
        x = self.avgpool(x).squeeze()
        #x is now (batch_size, embed_dim), reshape to (batch_size,␣
↪embed_dim*seq_length)
        x = x.unsqueeze(1).repeat(1, sequence_length, 1).reshape(batch_size, -1)
        #print (x.shape)
        x = self.fc(x)
        return x
```

Use the same training code as the one from part 1 to train your model. You may copy the training loop here. Expect to see close to `~90+%` test accuracy.

[ ]:
```
# Same training code

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms


# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyper-parameters
sequence_length = 28
input_size = 28
hidden_size = 64
num_layers = 2
num_classes = 10
num_epochs = 8
learning_rate = 0.005

# Initialize model
model = myModel(input_size=input_size, embed_dim=hidden_size,␣
↪seq_length=sequence_length)
model = model.to(device)

# Loss and optimizer
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()

        optimizer.step()

        if (i+1) % 10 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))


# Test the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Test Accuracy of the model on the 10000 test images: {} %'.
 ↪format(100 * correct / total))
```

```
Epoch [1/8], Step [10/157], Loss: 2.2828
Epoch [1/8], Step [20/157], Loss: 2.1009
Epoch [1/8], Step [30/157], Loss: 2.0814
Epoch [1/8], Step [40/157], Loss: 2.0351
Epoch [1/8], Step [50/157], Loss: 1.7069
Epoch [1/8], Step [60/157], Loss: 1.5384
Epoch [1/8], Step [70/157], Loss: 1.2281
```

```
Epoch [1/8], Step [80/157], Loss: 1.2421
Epoch [1/8], Step [90/157], Loss: 0.9071
Epoch [1/8], Step [100/157], Loss: 1.0152
Epoch [1/8], Step [110/157], Loss: 1.0221
Epoch [1/8], Step [120/157], Loss: 0.7912
Epoch [1/8], Step [130/157], Loss: 0.9861
Epoch [1/8], Step [140/157], Loss: 0.9273
Epoch [1/8], Step [150/157], Loss: 0.3981
Epoch [2/8], Step [10/157], Loss: 0.5219
Epoch [2/8], Step [20/157], Loss: 0.3848
Epoch [2/8], Step [30/157], Loss: 0.4828
Epoch [2/8], Step [40/157], Loss: 0.3515
Epoch [2/8], Step [50/157], Loss: 0.3199
Epoch [2/8], Step [60/157], Loss: 0.4379
Epoch [2/8], Step [70/157], Loss: 0.2631
Epoch [2/8], Step [80/157], Loss: 0.6981
Epoch [2/8], Step [90/157], Loss: 0.3145
Epoch [2/8], Step [100/157], Loss: 0.0908
Epoch [2/8], Step [110/157], Loss: 0.5518
Epoch [2/8], Step [120/157], Loss: 0.1450
Epoch [2/8], Step [130/157], Loss: 0.2894
Epoch [2/8], Step [140/157], Loss: 0.4532
Epoch [2/8], Step [150/157], Loss: 0.4064
Epoch [3/8], Step [10/157], Loss: 0.3476
Epoch [3/8], Step [20/157], Loss: 0.2695
Epoch [3/8], Step [30/157], Loss: 0.2704
Epoch [3/8], Step [40/157], Loss: 0.1873
Epoch [3/8], Step [50/157], Loss: 0.3324
Epoch [3/8], Step [60/157], Loss: 0.2485
Epoch [3/8], Step [70/157], Loss: 0.1246
Epoch [3/8], Step [80/157], Loss: 0.2628
Epoch [3/8], Step [90/157], Loss: 0.1256
Epoch [3/8], Step [100/157], Loss: 0.2612
Epoch [3/8], Step [110/157], Loss: 0.1905
Epoch [3/8], Step [120/157], Loss: 0.3247
Epoch [3/8], Step [130/157], Loss: 0.2417
Epoch [3/8], Step [140/157], Loss: 0.2004
Epoch [3/8], Step [150/157], Loss: 0.2540
Epoch [4/8], Step [10/157], Loss: 0.3945
Epoch [4/8], Step [20/157], Loss: 0.3144
Epoch [4/8], Step [30/157], Loss: 0.1910
Epoch [4/8], Step [40/157], Loss: 0.1318
Epoch [4/8], Step [50/157], Loss: 0.0313
Epoch [4/8], Step [60/157], Loss: 0.4591
Epoch [4/8], Step [70/157], Loss: 0.0772
Epoch [4/8], Step [80/157], Loss: 0.1102
Epoch [4/8], Step [90/157], Loss: 0.0971
Epoch [4/8], Step [100/157], Loss: 0.1128
```

```
Epoch [4/8], Step [110/157], Loss: 0.0593
Epoch [4/8], Step [120/157], Loss: 0.0651
Epoch [4/8], Step [130/157], Loss: 0.3856
Epoch [4/8], Step [140/157], Loss: 0.2799
Epoch [4/8], Step [150/157], Loss: 0.3647
Epoch [5/8], Step [10/157], Loss: 0.2450
Epoch [5/8], Step [20/157], Loss: 0.2590
Epoch [5/8], Step [30/157], Loss: 0.2107
Epoch [5/8], Step [40/157], Loss: 0.2014
Epoch [5/8], Step [50/157], Loss: 0.2095
Epoch [5/8], Step [60/157], Loss: 0.2476
Epoch [5/8], Step [70/157], Loss: 0.0805
Epoch [5/8], Step [80/157], Loss: 0.0447
Epoch [5/8], Step [90/157], Loss: 0.0669
Epoch [5/8], Step [100/157], Loss: 0.3559
Epoch [5/8], Step [110/157], Loss: 0.3194
Epoch [5/8], Step [120/157], Loss: 0.4159
Epoch [5/8], Step [130/157], Loss: 0.1283
Epoch [5/8], Step [140/157], Loss: 0.3808
Epoch [5/8], Step [150/157], Loss: 0.0932
Epoch [6/8], Step [10/157], Loss: 0.1420
Epoch [6/8], Step [20/157], Loss: 0.1451
Epoch [6/8], Step [30/157], Loss: 0.2559
Epoch [6/8], Step [40/157], Loss: 0.2276
Epoch [6/8], Step [50/157], Loss: 0.2416
Epoch [6/8], Step [60/157], Loss: 0.1026
Epoch [6/8], Step [70/157], Loss: 0.3104
Epoch [6/8], Step [80/157], Loss: 0.1303
Epoch [6/8], Step [90/157], Loss: 0.3692
Epoch [6/8], Step [100/157], Loss: 0.2997
Epoch [6/8], Step [110/157], Loss: 0.1554
Epoch [6/8], Step [120/157], Loss: 0.4348
Epoch [6/8], Step [130/157], Loss: 0.0989
Epoch [6/8], Step [140/157], Loss: 0.2593
Epoch [6/8], Step [150/157], Loss: 0.2819
Epoch [7/8], Step [10/157], Loss: 0.4255
Epoch [7/8], Step [20/157], Loss: 0.0821
Epoch [7/8], Step [30/157], Loss: 0.2576
Epoch [7/8], Step [40/157], Loss: 0.0421
Epoch [7/8], Step [50/157], Loss: 0.3974
Epoch [7/8], Step [60/157], Loss: 0.3511
Epoch [7/8], Step [70/157], Loss: 0.0553
Epoch [7/8], Step [80/157], Loss: 0.2630
Epoch [7/8], Step [90/157], Loss: 0.1038
Epoch [7/8], Step [100/157], Loss: 0.1796
Epoch [7/8], Step [110/157], Loss: 0.0677
Epoch [7/8], Step [120/157], Loss: 0.0944
Epoch [7/8], Step [130/157], Loss: 0.5563
```

```
Epoch [7/8], Step [140/157], Loss: 0.1437
Epoch [7/8], Step [150/157], Loss: 0.1907
Epoch [8/8], Step [10/157], Loss: 0.0535
Epoch [8/8], Step [20/157], Loss: 0.1444
Epoch [8/8], Step [30/157], Loss: 0.0460
Epoch [8/8], Step [40/157], Loss: 0.1303
Epoch [8/8], Step [50/157], Loss: 0.0268
Epoch [8/8], Step [60/157], Loss: 0.1730
Epoch [8/8], Step [70/157], Loss: 0.1698
Epoch [8/8], Step [80/157], Loss: 0.1753
Epoch [8/8], Step [90/157], Loss: 0.1119
Epoch [8/8], Step [100/157], Loss: 0.0360
Epoch [8/8], Step [110/157], Loss: 0.1577
Epoch [8/8], Step [120/157], Loss: 0.1090
Epoch [8/8], Step [130/157], Loss: 0.0034
Epoch [8/8], Step [140/157], Loss: 0.1164
Epoch [8/8], Step [150/157], Loss: 0.1692
Test Accuracy of the model on the 10000 test images: 95.16 %
```