Lab2

C2: Time measurement of code in C1

I will report average for each epochhere

	Epoch	CPU	GPU
data loading time	1	0.025910615921020508	0.3932163715362549
avg_mini batch train time	1	9.8288254737854	0.03323054313659668
total train time	1	3489.235363455451	15.246890783309937
data loading time	2	0.0238745893578932758	0.2598389538958939
avg_mini batch train time	2	8.874441146850586	0.01918935775756836
total train time	2	3463.2745983785978	15.200483560562134
data loading time	3	0.0195325238909074739	0.35345793279589039
avg_mini batch train time	3	9.043927907943726	0.02310657501220703
total train time	3	3524.34958932758982	15.430280685424805
data loading time	4	0.0234535789027583769	0.33334927573827834
avg_mini batch train time	4	8.815263509750366	0.019117116928100586
total train time	4	3357.35793278957328	15.497823953628549
data loading time	5	0.0234756802378239634	0.37238975923782589
avg_mini batch train time	5	9.046538352966309	0.0197908878326416
total train time	5	3359.37539275893299	15.517560005187988

C3: I/O optimization starting from code in C2

From the table, we can see that when num_workers = 0, we have the best performance

num_workers	Time
0	**0.0018935203552246094**
2	0.0020720958709716797
4	0.017673969268798828
8	0.06022214889526367
12	0.03252077102661133
16	0.051570892333984375

C4: Profiling starting from code in C3

When I use num_workers = 0, the time is 0.0018639564514160156. When I use num_workers = 1, the time is 0.0023262500762939453. From C3 and C4, we know that it does not necessary mean that increasing the num_workers can speed up the data loading. When we set to 0, it means the loader loads the data inside the main process. But if we have multiple cores, then num_workers can actually speed up.

C5: Training in GPUs V.S. CPUs

Epoch	CPU	GPU
1	3489.235363455451	15.246890783309937
2	3590.923850192859	15.200483560562134
3	3497.023579835923	15.430280685424805
4	3399.326952395230	15.497823953628549
5	3492.235983759893	15.517560005187988

C6: Experimenting with different optimizers

Optimizers	epoch	time	train loss	train top-1 acc
SGD	1	15.246890783309937	1.988328962069948	20.432928652711716
SGD	2	15.200483560562134	1.5172575635983205	41.66062737103774
SGD	3	15.430280685424805	1.2595373507960679	52.627654280262696
SGD	4	15.49782395362854	1.0342020011314041	61.37652780733207
SGD	5	15.517560005187988	0.8647981308915121	68.19055114819494
SGD_nes	1	28.066128730773926	2.034897204860092	21.97610816966551
SGD_nes	2	27.760812282562256	1.4966033151387559	42.179984230383624
SGD_nes	3	27.782509565353394	1.2289859423856906	53.260814863356245
SGD_nes	4	27.850111961364746	1.0070519291836282	62.75977421860316
SGD_nes	5	27.89001965522766	0.8746370840865327	68.43943285593723
Adagrad	1	27.82266616821289	2.2077978568918564	20.429481381731158
Adagrad	2	27.819855451583862	1.654606541709217	36.11329664655515
Adagrad	3	27.813787937164307	1.401751711850276	45.41344120363851
Adagrad	4	27.76784348487854	1.1763134882273272	55.15370209333665
Adagrad	5	27.80567979812622	1.035776418493227	61.52070716787579
Adadelta	1	28.552342891693115	1.368905180707917	38.71571214083069
Adadelta	2	28.53111982345581	0.8550770942817258	67.03340301211422
Adadelta	3	28.53568124771118	0.6677756459664201	75.50370191827194
Adadelta	4	28.60840344429016	0.5646420402447586	80.14308876379626
Adadelta	5	28.68533706665039	0.4976608041302322	82.27434027954197
Adam	1	15.157033205032349	1.8820709124245607	22.925146263828232
Adam	2	14.685014009475708	1.387173262093683	44.936456226266635
Adam	3	14.631178617477417	1.1834112846333047	55.79855950928615
Adam	4	14.67825698852539	1.0820588461883234	60.312539668888135
Adam	5	14.682486057281494	1.0340378732632494	62.93866078717115

C7: Experimenting without Batch Norm layer

We can actually notice that without BN layers, the accuracy drops.

with/without	epoch	time	train loss	train top-1 acc
With BN	1	15.246890783309937	1.988328962069948	20.432928652711716
With BN	2	15.200483560562134	1.5172575635983205	41.66062737103774
With BN	3	15.430280685424805	1.2595373507960679	52.627654280262696
With BN	4	15.49782395362854	1.0342020011314041	61.37652780733207
With BN	5	15.517560005187988	0.8647981308915121	68.19055114819494
Without BN	1	15.375982375892377	2.0021616421697086	16.277772587272928
Without BN	2	15.38572385372763	1.580125553223788	38.2379579586703
Without BN	3	15.98235783265723	1.3343231787766947	49.750428660848755
Without BN	4	15.28935732857923	1.1437207143325026	58.1836952755709
Without BN	5	15.23895782357878	1.0005435141760979	63.409153227689345

Q1

How many convolutional layers are in the ResNet-18 model ?

The structure is below. We have 17 convolutional layers and 1 FC layer. We have 3 projection layer too which are 1x1 convolutional layer

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
```

```
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
```

```
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (linear): Linear(in_features=512, out_features=10, bias=True)
)
```

Q2

What is the input dimension of the last linear layer?

512 * 4 * 4 = 8192 if we use the (3, 32, 32) image size

Q3

How many trainable parameters and how many gradients are in the ResNet-18 model that you build (please show both the answer and the code that you use to count them) when using the SGD optimizer.

There are 11173962 trainable parameters

```
from torchsummary import summary
summary(net, (3,32,32))
```

Below is the result

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
            Conv2d-3           [-1, 64, 32, 32]          36,864
       BatchNorm2d-4           [-1, 64, 32, 32]             128
            Conv2d-5           [-1, 64, 32, 32]          36,864
       BatchNorm2d-6           [-1, 64, 32, 32]             128
        BasicBlock-7           [-1, 64, 32, 32]               0
            Conv2d-8           [-1, 64, 32, 32]          36,864
       BatchNorm2d-9           [-1, 64, 32, 32]             128
           Conv2d-10           [-1, 64, 32, 32]          36,864
      BatchNorm2d-11           [-1, 64, 32, 32]             128
       BasicBlock-12           [-1, 64, 32, 32]               0
           Conv2d-13          [-1, 128, 16, 16]          73,728
      BatchNorm2d-14          [-1, 128, 16, 16]             256
           Conv2d-15          [-1, 128, 16, 16]         147,456
      BatchNorm2d-16          [-1, 128, 16, 16]             256
           Conv2d-17          [-1, 128, 16, 16]           8,192
      BatchNorm2d-18          [-1, 128, 16, 16]             256
       BasicBlock-19          [-1, 128, 16, 16]               0
           Conv2d-20          [-1, 128, 16, 16]         147,456
      BatchNorm2d-21          [-1, 128, 16, 16]             256
           Conv2d-22          [-1, 128, 16, 16]         147,456
      BatchNorm2d-23          [-1, 128, 16, 16]             256
       BasicBlock-24          [-1, 128, 16, 16]               0
           Conv2d-25            [-1, 256, 8, 8]         294,912
      BatchNorm2d-26            [-1, 256, 8, 8]             512
           Conv2d-27            [-1, 256, 8, 8]         589,824
      BatchNorm2d-28            [-1, 256, 8, 8]             512
           Conv2d-29            [-1, 256, 8, 8]          32,768
      BatchNorm2d-30            [-1, 256, 8, 8]             512
       BasicBlock-31            [-1, 256, 8, 8]               0
           Conv2d-32            [-1, 256, 8, 8]         589,824
      BatchNorm2d-33            [-1, 256, 8, 8]             512
           Conv2d-34            [-1, 256, 8, 8]         589,824
      BatchNorm2d-35            [-1, 256, 8, 8]             512
       BasicBlock-36            [-1, 256, 8, 8]               0
           Conv2d-37            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-38            [-1, 512, 4, 4]           1,024
           Conv2d-39            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-40            [-1, 512, 4, 4]           1,024
           Conv2d-41            [-1, 512, 4, 4]         131,072
      BatchNorm2d-42            [-1, 512, 4, 4]           1,024
       BasicBlock-43            [-1, 512, 4, 4]               0
           Conv2d-44            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-45            [-1, 512, 4, 4]           1,024
           Conv2d-46            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-47            [-1, 512, 4, 4]           1,024
       BasicBlock-48            [-1, 512, 4, 4]               0
           Linear-49                  [-1, 10]           5,130
================================================================
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
----------------------------------------------------------------
```

For SGD, it will happen 11173962 gradients because we have 11173962 learnable parameters

Q4

The same question as Q3, except now use Adam

When use Adam, it will has 11173962 gradients. Same as above.