C1

The DataLoader and ArgumentParser code is below

```
## ...

# argument parser
parser = argparse.ArgumentParser(description='cifar10 res18')
parser.add_argument('--lr', default=0.1, type=float, help='learning rate')
parser.add_argument('--workers', default=2, type=int, help='data loader workers number')
parser.add_argument('--epoch', default=5, type=int, help='epoch to run')
parser.add_argument('--data', default='./data', type=str, help='data path')
parser.add_argument('--resume', '-r', action='store_true', help='resume from checkpoint')
parser.add_argument('--gpu', action='store_true', help='use gpu or not')
parser.add_argument('--disable_batch_norm', action='store_true', help='use batch_norm or not')
# sgd sgd-nesterov adagrad adadelta adam
parser.add_argument('--optimizer', default='sgd', type=str, help='optimizer')
args = parser.parse_args()
device = 'cuda' if args.gpu and torch.cuda.is_available() else 'cpu'
data_path = args.data
workers_num = args.workers
max_epoch = args.epoch
disable_batch_norm = args.disable_batch_norm
lr = args.lr

# transformer
transform_train = transforms.Compose([
  transforms.RandomCrop(32, padding=4), # size of 32*32, padding 4 px
  transforms.RandomHorizontalFlip(),
  transforms.ToTensor(),
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
  transforms.ToTensor(),
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

# dataloaders
trainset = torchvision.datasets.CIFAR10(root=data_path, train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=workers_num)
```

```
testset = torchvision.datasets.CIFAR10(root=data_path, train=False,
download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100,
shuffle=False, num_workers=workers_num)


## ...
```

C2

Epoch	Data-loading	Training	Total Epoch
0	7.197	10.776	39.177
1	5.871	9.871	37.872
2	6.554	9.740	38.195
3	5.822	9.787	37.929
4	6.433	9.628	38.293
Total	31.877	49.802	191.466
Avg.	6.3754	9.9604	38.2932

C3

C3.1

Workers	Total Epoch
0	143.366
4	137.602
8	208.351
12	206.772
16	210.551

C3.2

4 workers is the best,

C4

Computing (Total run time)

Epoch	workers=1	workers=4
0	40.454	27.991
1	39.362	27.271
2	39.324	27.266
3	39.150	27.370
4	39.338	27.705
Total	197.628	137.602
Avg.	39.5256	27.5204

I guess the reason is that though more workers can load data in parallel, too many worker might cost too much resources on scheduling or something else.

C5

wokers = 4

Epoch	GPU	CPU
0	39.177	711.004
1	37.872	739.601
2	38.195	773.454
3	37.929	770.572
4	38.293	861.887
Total	191.466	3856.519
Avg.	38.2932	771.3038

(CPU mode was torturing....)

C6

Optimizer	Avg. Training Time	Loss	Accuracy
SGD	26.573	0.887	68.532%
SGD with nesterov	26.9978	0.784	72.342%
Adagrad	27.3416	1.216	55.704%
Adadelta	41.8614	0.505	82.412%
Adam	39.821	1.777	31.934%

we can see that:

1) sgd and its variant is the fastest among optimizers.

2) adadelta is the slowest but achieves the best accuracy.

C7

Optimizer	Avg. Training Time	Loss	Accuracy
SGD-no-bn	34.835	0.988	64.062%
SGD-bn	26.573	0.887	68.532%

The result is not quite stable, because the accuracy is always different. Sometimes, SGD_without bn can even have accuracy of 74%! I don't know why the figure changes so dramatically.

Q1

If we ignore the 1×1 conv layer:

Then: $1 + 4 \times 2 \times 2 = 17$

Q2

512

Q3

```
pytorch_trainable_total_params = sum(p.numel() for p in net.parameters() if p.requires_grad)
print('Trainable parameters: %s' %(pytorch_trainable_total_params))
```

Trainable parameters: 11173962

Gradients: 11173962

They two are identical in SGD.

But we can also calculate it based on ResNet-18 model structure.

Q4

The parameters remains the same: 11173962

And I presume "gradients" in the question means how many times we use gradients in one step. If so, my anwser would be three times of params because Adam needs previous gradients (momentum v_{t-1} actually) and current gradient (g_t) and element-wise square $g_t \odot g_t$. Thus 3× parameters.