Python与深度学习基础

第二次作业实验报告

母子跃 PB20000010

一、 在Tiny-ImageNet数据集上训练Resnet模型

注：本次作业使用的是Bitahub中自带的Tiny-ImageNet数据集，数据量较小

(1)根据Tiny-ImageNet图片大小（3*64*64），计算图片经过各层处理后的中间结果的大小。请列出各层的名称及输出的大小。

输入图片大小：3×64×64

第一层卷积处理：Conv1，卷积核大小为7×7，步长为2，填充为3，输出特征图尺寸为：64×32×32

第二层池化处理：Maxpool，池化核大小为3×3，步长为2，填充为1，输出特征图尺寸为：64×16×16

第一个残差块：layer1，包含两个基本块（BasicBlock），需要分别计算两个基本块的输出：

1）第一个基本块：输出特征图尺寸为64×16×16。

2）第二个基本块：输出特征图尺寸为64×16×16。

第二个残差块：layer2，包含两个基本块，同样需要分别计算两个基本块的输出：

1）第一个基本块：输出特征图尺寸为128×8×8。

2）第二个基本块：输出特征图尺寸为128×8×8。

第三个残差块：layer3，包含两个基本块，同样需要分别计算两个基本块的输出：

1）第一个基本块：输出特征图尺寸为256×4×4。

2）第二个基本块：输出特征图尺寸为256×4×4。

第四个残差块：layer4，包含两个基本块，同样需要分别计算两个基本块的输出：

1）第一个基本块：输出特征图尺寸为512×2×2。

2）第二个基本块：输出特征图尺寸为512×2×2。

全局平均池化层：Avgpool，池化核大小为2×2，步长为1，输出特征图尺寸为：512×1×1。

全连接层：Fc，将输入展开为一维向量，输出维度为200。

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           9,408
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
         MaxPool2d-4           [-1, 64, 16, 16]               0
            Conv2d-5           [-1, 64, 16, 16]          36,864
       BatchNorm2d-6           [-1, 64, 16, 16]             128
              ReLU-7           [-1, 64, 16, 16]               0
```

Conv2d-8	[-1, 64, 16, 16]	36,864
BatchNorm2d-9	[-1, 64, 16, 16]	128
ReLU-10	[-1, 64, 16, 16]	0
BasicBlock-11	[-1, 64, 16, 16]	0
Conv2d-12	[-1, 64, 16, 16]	36,864
BatchNorm2d-13	[-1, 64, 16, 16]	128
ReLU-14	[-1, 64, 16, 16]	0
Conv2d-15	[-1, 64, 16, 16]	36,864
BatchNorm2d-16	[-1, 64, 16, 16]	128
ReLU-17	[-1, 64, 16, 16]	0
BasicBlock-18	[-1, 64, 16, 16]	0
Conv2d-19	[-1, 128, 8, 8]	73,728
BatchNorm2d-20	[-1, 128, 8, 8]	256
ReLU-21	[-1, 128, 8, 8]	0
Conv2d-22	[-1, 128, 8, 8]	147,456
BatchNorm2d-23	[-1, 128, 8, 8]	256
Conv2d-24	[-1, 128, 8, 8]	8,192
BatchNorm2d-25	[-1, 128, 8, 8]	256
ReLU-26	[-1, 128, 8, 8]	0
BasicBlock-27	[-1, 128, 8, 8]	0
Conv2d-28	[-1, 128, 8, 8]	147,456
BatchNorm2d-29	[-1, 128, 8, 8]	256
ReLU-30	[-1, 128, 8, 8]	0
Conv2d-31	[-1, 128, 8, 8]	147,456
BatchNorm2d-32	[-1, 128, 8, 8]	256
ReLU-33	[-1, 128, 8, 8]	0
BasicBlock-34	[-1, 128, 8, 8]	0
Conv2d-35	[-1, 256, 4, 4]	294,912
BatchNorm2d-36	[-1, 256, 4, 4]	512
ReLU-37	[-1, 256, 4, 4]	0
Conv2d-38	[-1, 256, 4, 4]	589,824
BatchNorm2d-39	[-1, 256, 4, 4]	512
Conv2d-40	[-1, 256, 4, 4]	32,768
BatchNorm2d-41	[-1, 256, 4, 4]	512
ReLU-42	[-1, 256, 4, 4]	0
BasicBlock-43	[-1, 256, 4, 4]	0
Conv2d-44	[-1, 256, 4, 4]	589,824
BatchNorm2d-45	[-1, 256, 4, 4]	512
ReLU-46	[-1, 256, 4, 4]	0
Conv2d-47	[-1, 256, 4, 4]	589,824
BatchNorm2d-48	[-1, 256, 4, 4]	512
ReLU-49	[-1, 256, 4, 4]	0
BasicBlock-50	[-1, 256, 4, 4]	0
Conv2d-51	[-1, 512, 2, 2]	1,179,648

```
      BatchNorm2d-52            [-1, 512, 2, 2]           1,024
             ReLU-53            [-1, 512, 2, 2]               0
           Conv2d-54            [-1, 512, 2, 2]       2,359,296
      BatchNorm2d-55            [-1, 512, 2, 2]           1,024
           Conv2d-56            [-1, 512, 2, 2]         131,072
      BatchNorm2d-57            [-1, 512, 2, 2]           1,024
             ReLU-58            [-1, 512, 2, 2]               0
       BasicBlock-59            [-1, 512, 2, 2]               0
           Conv2d-60            [-1, 512, 2, 2]       2,359,296
      BatchNorm2d-61            [-1, 512, 2, 2]           1,024
             ReLU-62            [-1, 512, 2, 2]               0
           Conv2d-63            [-1, 512, 2, 2]       2,359,296
      BatchNorm2d-64            [-1, 512, 2, 2]           1,024
             ReLU-65            [-1, 512, 2, 2]               0
       BasicBlock-66            [-1, 512, 2, 2]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                  [-1, 200]         102,600
================================================================
Total params: 11,279,112
Trainable params: 11,279,112
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 5.13
Params size (MB): 43.03
Estimated Total Size (MB): 48.20
```

（2）原代码中对应的是 ImageNet，它有 1000 类，也就是 output 有 1000 维，修改成 200 维：

```
# ----------change: use function 'torch.nn.Linear()' to change the output dimension of the fully connected layer
out_features = 200                     # output features for tiny-imagenet
in_features = model.fc.in_features     # the original features of resnet
model.fc = torch.nn.Linear(in_features, out_features)
```

（3）处理数据集，修改正确标签

首先，设置 traindir 变量为 Tiny-ImageNet 训练集文件夹的路径；
然后，通过打开 val_annotations.txt 文件并读取其中的标签信息，将每个图像与其对应的标签建立映射关系，并存储在字典 d 中；
接下来，根据字典 d 的内容，为每个类别创建一个新的文件夹，用于存储 Tiny-ImageNet 验证集中该类别的所有图像；
对于每张图像，从原始 Tiny-ImageNet 验证集中读取其图像数据，并将其复制到相应的新文件夹中；
最后，设置 valdir 变量为 Tiny-ImageNet 新建的验证集文件夹的路径，并进行数据归一化处理。

```python
# Data Loading code
# -------------- Change: We need to change the construction in valdir when using tiny-imagenet ------------#
traindir = os.path.join(args.data, 'train')
# valdir = os.path.join(args.data, 'val')
f = open("/data/bitahub/Tiny-ImageNet/val/val_annotations.txt","r")       # the relative path of the file is corresponding to local path
val_labels = f.readlines()      # get labels in val_annotations.txt to correct the original labels
f.close()

d = {}        # create a new dictionary to store the map between images and labels
for item in val_labels:
    # split according to the format of val_annotations.txt
    image_name = item.split("\t")[0]
    image_label = item.split("\t")[1]
    if image_label not in d:
        d[image_label] = [image_name]
    else:
        d[image_label] += [image_name]
# for item in d:
#     if not os.path.exists("/data/bitahub/Tiny-ImageNet/my_val/{}/images".format(item)):
#         os.makedirs("/data/bitahub/Tiny-ImageNet/my_val/{}/images".format(item))
#         # os.makedirs() could only been used when the path is not existed
#     for num,img in enumerate(d[item]):
#         source = "/data/bitahub/Tiny-ImageNet/val/images/{}".format(img)
#         destination = "/data/bitahub/Tiny-ImageNet/my_val/{}/images/{}_{}.JPEG".format(item, item, num)
#         # using shutil.copyfile() to create new val_dir just like the train_dir
#         shutil.copyfile(source, destination)

# the dataset on bitahub is "read-only" mode, so cann't write the new folder "my_val"
for item in d:
    if not os.path.exists("/mydata/my_val/{}/images".format(item)):
        os.makedirs("/mydata/my_val/{}/images".format(item))
        # os.makedirs() could only been used when the path is not existed
    for num,img in enumerate(d[item]):
        source = "/data/bitahub/Tiny-ImageNet/val/images/{}".format(img)
        destination = "/mydata/my_val/{}/images/{}_{}.JPEG".format(item, item, num)
        # using shutil.copyfile() to create new val_dir just like the train_dir
        shutil.copyfile(source, destination)
# create new val_dir
# valdir = os.path.join(args.data, 'my_val')
valdir = '/mydata/my_val'
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

（4）在代码中增加 torch.utils.tensorboard 的代码，以能在 TensorBoard 中观察训练集 Loss、训练集精度、验证集 Loss、验证集精度的变化。

首先，代码使用 datetime.now().strftime() 函数获取当前时间，并且将其作为 tensorboard 存储的目录。之后，将训练数据加载器传递给 train() 函数，获得返回的 train_loss 和 train_acc 值。同时，将验证数据加载器传递给 validate() 函数，获得返回的 val_loss、acc1 和 val_acc 值。这些值将用于绘制 scalar（标量）图，以便能够更好地理解模型的训练和验证过程。

在每个 epoch 结束时，将会检查当前是否是最佳的模型，并在 args.save_path 路径下保存模型参数。在第 5 和 10 个 epoch 结束时，也会将模型参数保存在 args.save_path 路径下，以便查看模型在不同阶段的表现。

最后，在 TensorBoard 中绘制 scalar 图，记录训练 loss、训练 acc、验证 loss 和验证 acc 的变化情况。在训练过程结束后，关闭 TensorBoard 的写入器 writer，输出 "Training Finished"。

```python
#--------------- Change: no need to reshape and flip in tiny-imagenet ---------------------#
val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=False,
    num_workers=args.workers, pin_memory=True)

# Create Tensorboard and assign its storage dir as "datetime + name_of_web_construction"
current_time = datetime.now().strftime('%b%d_%H-%M-%S')
logdir = os.path.join('/output', 'logs', current_time + '_' + args.arch)
writer = SummaryWriter(logdir)

# Use Tendorboard to draw Graph of the net
# dummy_input = torch.rand(4, 3, 64, 64)          # The corresponding input of tiny-imagenet
# writer.add_graph(model, (dummy_input,))       # model in this .py file cannot be drawn by this way,
summary(model, (3, 64, 64))         # Instead, using lib "summary" to represent the construction

if args.evaluate:
    validate(val_loader, model, criterion, args)
    return

for epoch in range(args.start_epoch, args.epochs):
    if args.distributed:
        train_sampler.set_epoch(epoch)

    # train for one epoch
    # -------------Change: We need more returned value to draw tensorboard--------------------#
    train_loss, train_acc = train(train_loader, model, criterion, optimizer, epoch, args)

    # evaluate on validation set
    acc1, val_loss, val_acc = validate(val_loader, model, criterion, args)

    scheduler.step()


    # remember best acc@1 and save checkpoint
    is_best = acc1 > best_acc1
    best_acc1 = max(acc1, best_acc1)

    if not args.multiprocessing_distributed or (args.multiprocessing_distributed
            and args.rank % ngpus_per_node == 0):
        save_checkpoint({
            'epoch': epoch + 1,
            'arch': args.arch,
            'state_dict': model.state_dict(),
            'best_acc1': best_acc1,
            'optimizer' : optimizer.state_dict(),
            'scheduler' : scheduler.state_dict()
        }, is_best)
    # pick another two checkpoints for evaluation
    if epoch == 5 or epoch == 10:
         save_checkpoint({
            'epoch': epoch + 1,
            'arch': args.arch,
            'state_dict': model.state_dict(),
            'best_acc1': best_acc1,
            'optimizer' : optimizer.state_dict(),
            'scheduler' : scheduler.state_dict()
        }, is_best,"checkpoint_epoch{}.pth.tar".format(epoch))

    # when one epoch finished, save scalar in tensorboard for visibility
    writer.add_scalar('scalar/train_loss', train_loss, epoch)
    writer.add_scalar('scalar/train_acc', train_acc, epoch)
    writer.add_scalar('scalar/val_loss', val_loss, epoch)
    writer.add_scalar('scalar/val_acc', val_acc, epoch)

print("Training Finished")
writer.close()
```

在 train() 函数中，首先使用 AverageMeter 类初始化了 batch_time（每批数据的时间）、data_time（数据加载时间）、losses（损失）、top1（Top1 准确率）和 top5（Top5 准确率）五个度量器。之后，使用 ProgressMeter 类初始化了 progress，该类用于打印每个 epoch 的进度条。

在每个 epoch 开始时，将模型转换为训练模式 model.train()。对于每个 batch 数据，先记录数据加载所需时间 data_time，同时将数据 images 和标签 target 分别移动到 GPU 上（如果有）。

然后，计算模型的输出 output 和损失 loss，并通过 accuracy() 函数计算准确率 acc1 和 acc5。将损失值和准确率值记录在相应度量器中。

接着，清空优化器 optimizer 中的梯度信息 optimizer.zero_grad()，反向传播 loss.backward() 并更新模型参数 optimizer.step()。

最后，在每 args.print_freq（默认为 10） 个 batch 之后，打印当前进度 progress.display(i)。返回 loss 和 top5.avg（Top5 准确率的平均值）两个值。

```python
def train(train_loader, model, criterion, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses, top1, top5],
        prefix="Epoch: [{}]".format(epoch))

    # switch to train mode
    model.train()

    end = time.time()
    for i, (images, target) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)

        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        if torch.cuda.is_available():
            target = target.cuda(args.gpu, non_blocking=True)

        # compute output
        output = model(images)
        loss = criterion(output, target)

        # measure accuracy and record loss
        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        top5.update(acc5[0], images.size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()

        if i % args.print_freq == 0:
            progress.display(i)

    #--------------Change: Add more return values----------------
    return loss, top5.avg       # Our aim is to boost the average c
```

在 validate() 函数中，依然是使用 AverageMeter 类分别初始化 batch_time（每批数据的时间）、losses（损失）、top1（Top1 准确率）和 top5（Top5 准确率）四个度量器。需要注意的是，在 top1 和 top5 的初始化中，我们使用了 Summary 枚举类来指定每次更新后是否影响其总数值，以及如何汇总每个 batch 中的值。

接着，在进入“测试模式” model.eval() 后，使用 with torch.no_grad() 包含 for 循环，避免计算图的构建和梯度更新，从而减少 GPU 存储需求和计算量。

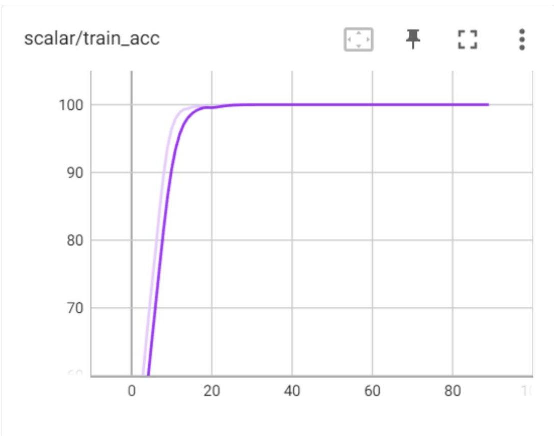
在每个 batch 数据上，与 train() 函数类似，将数据 images 和标签 target 分别移动到 GPU 上，并计算模型的输出 output 和损失 loss。使用 accuracy() 函数计算准确率 acc1 和 acc5，同时在相应度量器中记录损失值和准确率值。

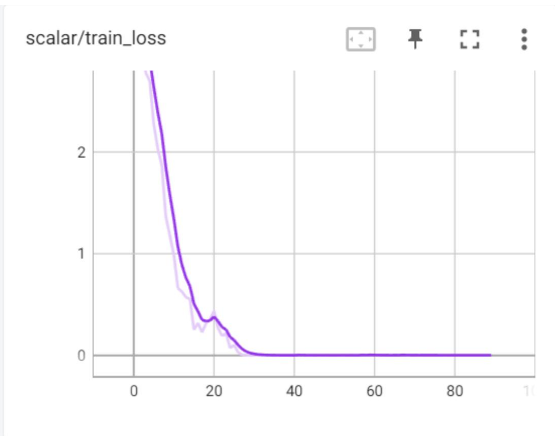
在计算完每个 batch 后，更新其他度量器 batch_time 和 progress。progress.display(i) 用于打印当前进度，如果 args.evaluate 为 True，则会额外打印出每张图片的预测结果和真实标签，最后调用 progress.display_summary() 打印所有 epoch 的总结信息。

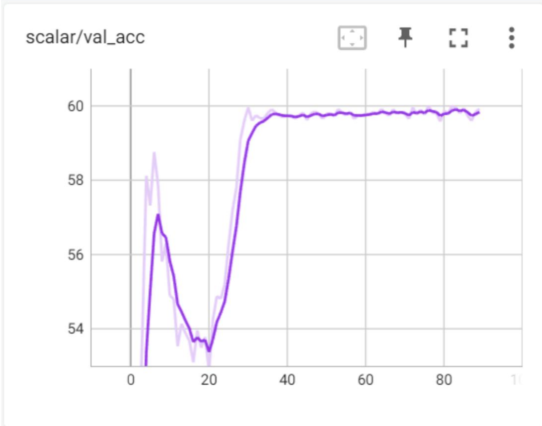
最后，返回 top1.avg（Top1 准确率的平均值）、loss（损失值）和 top5.avg（Top5 准确率的平均值）三个值。

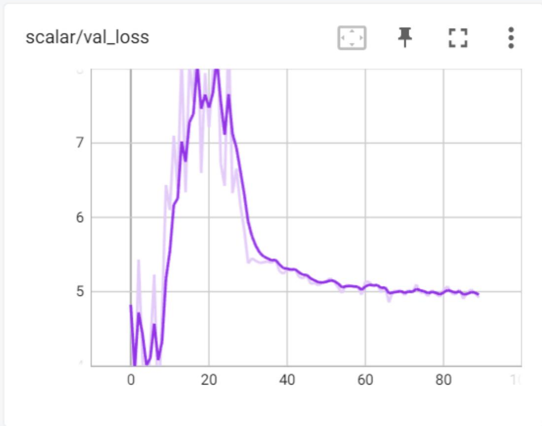
```python
def validate(val_loader, model, criterion, args):
    batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
    losses = AverageMeter('Loss', ':.4e', Summary.NONE)
    top1 = AverageMeter('Acc@1', ':6.2f', Summary.AVERAGE)
    top5 = AverageMeter('Acc@5', ':6.2f', Summary.AVERAGE)
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, losses, top1, top5],
        prefix='Test: ')

    # switch to evaluate mode
    model.eval()

    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            if args.gpu is not None:
                images = images.cuda(args.gpu, non_blocking=True)
            if torch.cuda.is_available():
                target = target.cuda(args.gpu, non_blocking=True)

            # compute output
            output = model(images)
            loss = criterion(output, target)

            # measure accuracy and record loss
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            losses.update(loss.item(), images.size(0))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))

            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()

            if i % args.print_freq == 0:
                progress.display(i)

            if args.evaluate:       # when evaluating, show more detail about each picture
                print("pic:{:2}  aim:{:2}  result:{:4} {:>8}".format(i, int(target), int(output.argmax()), 'correct' if int(target) == int(output.argmax()) else 'false'))
                if i == 100:        # no need to caculate all 10,000 pics
                    break

        progress.display_summary()

    #--------------Change: Add more return values----------------#
    return top1.avg, loss, top5.avg
```

（5）对第 3 步中的曲线进行截图

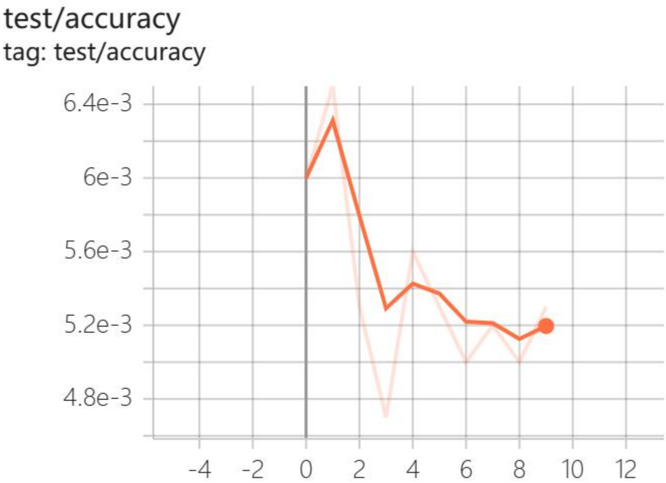








需要注意的是，在前几次的训练过程中，出现了如下图中的“过拟合”现象：



（6）分别在无 GPU、1 个 GPU、多个 GPU 环境下，重复上述过程，观察和量化评价训练速度上的差异

修改输入的 GPU 数量：

```
def main():
    args = argparse.Namespace(
        arch='resnet18',
        data_url=None,
        data_dir=None,
        workers=2,
        epochs=90,
        start_epoch=0,
        batch_size=256,
        lr=0.1,
        momentum=0.9,
        weight_decay=1e-4,
        print_freq=10,
        resume=None,
        world_size=1,
        rank=0,
        dist_url='tcp://127.0.0.1:23456',
        dist_backend='nccl',
        seed=42,
        gpu=8,#修改gpu个数
        pretrained=True,
        data='/data/bitahub/Tiny-ImageNet',
        evaluate=True,
        multiprocessing_distributed=False

    )
```

由于训练时间过长，导致无法统计出具体的训练时间

（7）保存 2 个训练过程中模型的 checkpoint

- checkpoint_epoch10.pth.tar
- checkpoint_epoch5.pth.tar
- checkpoint.pth.tar

（8）运行时部分过程截图

```
Epoch: [14][390/391]    Time  0.037 ( 0.180)    Data  0.000 ( 0.146)    Loss 5.5408e-01 (2.9483e-01)    Acc@1  85.00 ( 91.01)   Acc@5  96.25 ( 99.39)
Test: [ 0/40]   Time  0.447 ( 0.447)    Loss 3.4266e+00 (3.4266e+00)    Acc@1  41.02 ( 41.02)   Acc@5  69.53 ( 69.53)
Test: [10/40]   Time  0.015 ( 0.077)    Loss 5.4116e+00 (4.4403e+00)    Acc@1  23.44 ( 30.79)   Acc@5  48.83 ( 57.88)
Test: [20/40]   Time  0.100 ( 0.063)    Loss 4.3161e+00 (4.7367e+00)    Acc@1  30.08 ( 28.05)   Acc@5  55.47 ( 54.00)
Test: [30/40]   Time  0.015 ( 0.057)    Loss 5.3962e+00 (4.8278e+00)    Acc@1  19.92 ( 27.51)   Acc@5  47.27 ( 52.84)
 *   Acc@1 28.070 Acc@5 53.880
Epoch: [15][  0/391]    Time  1.084 ( 1.084)    Data  1.054 ( 1.054)    Loss 1.9262e-01 (1.9262e-01)    Acc@1  94.14 ( 94.14)   Acc@5 100.00 (100.00)
Epoch: [15][ 10/391]    Time  0.043 ( 0.226)    Data  0.000 ( 0.187)    Loss 2.3546e-01 (2.4337e-01)    Acc@1  93.75 ( 92.37)   Acc@5  99.61 ( 99.72)
Epoch: [15][ 20/391]    Time  0.564 ( 0.219)    Data  0.537 ( 0.181)    Loss 2.6244e-01 (2.4035e-01)    Acc@1  91.02 ( 92.76)   Acc@5  99.61 ( 99.72)
Epoch: [15][ 30/391]    Time  0.039 ( 0.198)    Data  0.000 ( 0.160)    Loss 3.2740e-01 (2.4668e-01)    Acc@1  90.23 ( 92.39)   Acc@5  99.22 ( 99.65)
Epoch: [15][ 40/391]    Time  0.553 ( 0.197)    Data  0.523 ( 0.159)    Loss 2.3604e-01 (2.4709e-01)    Acc@1  92.97 ( 92.44)   Acc@5 100.00 ( 99.66)
Epoch: [15][ 50/391]    Time  0.045 ( 0.189)    Data  0.011 ( 0.152)    Loss 2.0222e-01 (2.4366e-01)    Acc@1  92.19 ( 92.56)   Acc@5 100.00 ( 99.63)
Epoch: [15][ 60/391]    Time  0.495 ( 0.190)    Data  0.469 ( 0.153)    Loss 1.6566e-01 (2.3714e-01)    Acc@1  95.31 ( 92.78)   Acc@5 100.00 ( 99.67)
Epoch: [15][ 70/391]    Time  0.102 ( 0.185)    Data  0.074 ( 0.149)    Loss 1.4021e-01 (2.3240e-01)    Acc@1  96.09 ( 92.90)   Acc@5 100.00 ( 99.68)
Epoch: [15][ 80/391]    Time  0.536 ( 0.186)    Data  0.510 ( 0.150)    Loss 1.8382e-01 (2.2665e-01)    Acc@1  93.36 ( 93.14)   Acc@5  99.61 ( 99.70)
Epoch: [15][ 90/391]    Time  0.043 ( 0.182)    Data  0.011 ( 0.147)    Loss 2.3303e-01 (2.2299e-01)    Acc@1  92.58 ( 93.32)   Acc@5 100.00 ( 99.73)
Epoch: [15][100/391]    Time  0.475 ( 0.183)    Data  0.445 ( 0.149)    Loss 1.5376e-01 (2.1845e-01)    Acc@1  94.92 ( 93.49)   Acc@5 100.00 ( 99.74)
Epoch: [15][110/391]    Time  0.100 ( 0.182)    Data  0.072 ( 0.147)    Loss 1.9349e-01 (2.1447e-01)    Acc@1  94.14 ( 93.64)   Acc@5 100.00 ( 99.76)
Epoch: [15][120/391]    Time  0.495 ( 0.182)    Data  0.465 ( 0.148)    Loss 2.1741e-01 (2.1163e-01)    Acc@1  92.97 ( 93.71)   Acc@5 100.00 ( 99.78)
Epoch: [15][130/391]    Time  0.122 ( 0.180)    Data  0.094 ( 0.145)    Loss 2.1334e-01 (2.0835e-01)    Acc@1  92.58 ( 93.84)   Acc@5 100.00 ( 99.78)
Epoch: [15][140/391]    Time  0.420 ( 0.180)    Data  0.390 ( 0.146)    Loss 1.8477e-01 (2.0788e-01)    Acc@1  95.31 ( 93.83)   Acc@5  99.61 ( 99.77)
Epoch: [15][150/391]    Time  0.202 ( 0.179)    Data  0.170 ( 0.145)    Loss 2.2551e-01 (2.0700e-01)    Acc@1  94.92 ( 93.87)   Acc@5 100.00 ( 99.77)
Epoch: [15][160/391]    Time  0.474 ( 0.180)    Data  0.447 ( 0.145)    Loss 1.7385e-01 (2.0579e-01)    Acc@1  94.92 ( 93.89)   Acc@5 100.00 ( 99.77)
Epoch: [15][170/391]    Time  0.260 ( 0.180)    Data  0.236 ( 0.146)    Loss 2.3954e-01 (2.0525e-01)    Acc@1  91.41 ( 93.90)   Acc@5 100.00 ( 99.77)
Epoch: [15][180/391]    Time  0.408 ( 0.181)    Data  0.379 ( 0.147)    Loss 1.4985e-01 (2.0513e-01)    Acc@1  95.31 ( 93.91)   Acc@5 100.00 ( 99.77)
Epoch: [15][190/391]    Time  0.245 ( 0.180)    Data  0.217 ( 0.146)    Loss 1.8823e-01 (2.0439e-01)    Acc@1  95.31 ( 93.94)   Acc@5  99.22 ( 99.77)
Epoch: [15][200/391]    Time  0.419 ( 0.181)    Data  0.392 ( 0.147)    Loss 2.2993e-01 (2.0544e-01)    Acc@1  94.14 ( 93.89)   Acc@5  99.61 ( 99.77)
Epoch: [15][210/391]    Time  0.191 ( 0.181)    Data  0.164 ( 0.147)    Loss 2.8236e-01 (2.0604e-01)    Acc@1  90.62 ( 93.86)   Acc@5  99.61 ( 99.77)
Epoch: [15][220/391]    Time  0.420 ( 0.181)    Data  0.389 ( 0.147)    Loss 1.9513e-01 (2.0709e-01)    Acc@1  94.92 ( 93.85)   Acc@5  99.61 ( 99.76)
Epoch: [15][230/391]    Time  0.254 ( 0.181)    Data  0.223 ( 0.147)    Loss 2.8844e-01 (2.0834e-01)    Acc@1  91.02 ( 93.80)   Acc@5  99.61 ( 99.76)
Epoch: [15][240/391]    Time  0.407 ( 0.181)    Data  0.374 ( 0.147)    Loss 2.5334e-01 (2.1014e-01)    Acc@1  91.02 ( 93.74)   Acc@5  99.22 ( 99.76)
```

```
Epoch: [22][220/391]    Time  1.216 ( 0.615)    Data  1.085 ( 0.497)    Loss 2.7916e-01 (1.8800e-01)    Acc@1  90.23 ( 9
4.40)   Acc@5  99.61 ( 99.72)
Epoch: [22][230/391]    Time  0.104 ( 0.600)    Data  0.001 ( 0.482)    Loss 2.2466e-01 (1.9033e-01)    Acc@1  92.19 ( 9
4.33)   Acc@5  99.61 ( 99.70)
Epoch: [22][240/391]    Time  0.446 ( 0.585)    Data  0.309 ( 0.467)    Loss 3.1815e-01 (1.9245e-01)    Acc@1  90.23 ( 9
4.24)   Acc@5  98.83 ( 99.69)
Epoch: [22][250/391]    Time  0.120 ( 0.569)    Data  0.001 ( 0.451)    Loss 2.5826e-01 (1.9396e-01)    Acc@1  91.80 ( 9
4.22)   Acc@5  99.22 ( 99.69)
Epoch: [22][260/391]    Time  0.473 ( 0.555)    Data  0.339 ( 0.436)    Loss 3.1189e-01 (1.9627e-01)    Acc@1  89.45 ( 9
4.13)   Acc@5  99.22 ( 99.69)
Epoch: [22][270/391]    Time  0.111 ( 0.541)    Data  0.001 ( 0.423)    Loss 2.2380e-01 (1.9772e-01)    Acc@1  92.58 ( 9
4.08)   Acc@5  99.61 ( 99.69)
Epoch: [22][280/391]    Time  1.026 ( 0.536)    Data  0.894 ( 0.418)    Loss 2.8041e-01 (2.0047e-01)    Acc@1  91.41 ( 9
3.97)   Acc@5  98.83 ( 99.67)
Epoch: [22][290/391]    Time  0.109 ( 0.531)    Data  0.001 ( 0.413)    Loss 2.5815e-01 (2.0272e-01)    Acc@1  91.41 ( 9
3.90)   Acc@5  99.61 ( 99.67)
Epoch: [22][300/391]    Time  1.212 ( 0.530)    Data  1.087 ( 0.411)    Loss 2.8415e-01 (2.0528e-01)    Acc@1  91.41 ( 9
3.81)   Acc@5  98.83 ( 99.66)
Epoch: [22][310/391]    Time  0.121 ( 0.523)    Data  0.001 ( 0.404)    Loss 2.6021e-01 (2.0699e-01)    Acc@1  93.36 ( 9
3.77)   Acc@5  99.61 ( 99.65)
Epoch: [22][320/391]    Time  0.900 ( 0.515)    Data  0.651 ( 0.396)    Loss 2.6036e-01 (2.0927e-01)    Acc@1  91.02 ( 9
3.68)   Acc@5  99.22 ( 99.64)
Epoch: [22][330/391]    Time  0.113 ( 0.505)    Data  0.001 ( 0.386)    Loss 2.0334e-01 (2.1194e-01)    Acc@1  95.70 ( 9
3.61)   Acc@5  99.61 ( 99.63)
Epoch: [22][340/391]    Time  1.226 ( 0.499)    Data  1.108 ( 0.380)    Loss 2.7873e-01 (2.1462e-01)    Acc@1  91.41 ( 9
3.53)   Acc@5  99.61 ( 99.61)
Epoch: [22][350/391]    Time  0.111 ( 0.496)    Data  0.001 ( 0.377)    Loss 2.6511e-01 (2.1615e-01)    Acc@1  93.36 ( 9
3.49)   Acc@5 100.00 ( 99.61)
Epoch: [22][360/391]    Time  1.602 ( 0.500)    Data  1.485 ( 0.381)    Loss 2.8292e-01 (2.1853e-01)    Acc@1  89.84 ( 9
3.42)   Acc@5  99.61 ( 99.60)
Epoch: [22][370/391]    Time  0.110 ( 0.497)    Data  0.001 ( 0.378)    Loss 4.1160e-01 (2.2198e-01)    Acc@1  88.28 ( 9
3.30)   Acc@5  98.83 ( 99.59)
Epoch: [22][380/391]    Time  1.701 ( 0.501)    Data  1.580 ( 0.382)    Loss 4.0028e-01 (2.2427e-01)    Acc@1  86.33 ( 9
3.23)   Acc@5  99.22 ( 99.58)
Epoch: [22][390/391]    Time  0.104 ( 0.500)    Data  0.000 ( 0.382)    Loss 4.7682e-01 (2.2714e-01)    Acc@1  83.75 ( 9
3.14)   Acc@5  98.12 ( 99.57)
Test: [ 0/40]   Time  1.226 ( 1.226)    Loss 4.5341e+00 (4.5341e+00)    Acc@1  30.08 ( 30.08)   Acc@5  56.64 ( 56.64)
Test: [10/40]   Time  0.060 ( 0.191)    Loss 4.5721e+00 (4.5636e+00)    Acc@1  33.98 ( 30.15)   Acc@5  58.59 ( 55.58)
Test: [20/40]   Time  0.073 ( 0.135)    Loss 4.9465e+00 (4.8074e+00)    Acc@1  28.12 ( 28.12)   Acc@5  49.61 ( 52.57)
Test: [30/40]   Time  0.068 ( 0.116)    Loss 5.2082e+00 (4.8877e+00)    Acc@1  28.12 ( 28.23)   Acc@5  50.78 ( 51.80)
```

```
Epoch: [89][150/391]    Time  0.042 ( 0.175)    Data  0.000 ( 0.140)    Loss 2.5321e-03 (2.8378e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][160/391]    Time  0.215 ( 0.174)    Data  0.187 ( 0.140)    Loss 2.4638e-03 (2.8237e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][170/391]    Time  0.039 ( 0.174)    Data  0.000 ( 0.140)    Loss 2.6011e-03 (2.8371e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][180/391]    Time  0.306 ( 0.173)    Data  0.278 ( 0.139)    Loss 2.2875e-03 (2.8340e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][190/391]    Time  0.042 ( 0.173)    Data  0.000 ( 0.138)    Loss 5.8494e-03 (2.8573e-03)    Acc@1  99.61 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][200/391]    Time  0.265 ( 0.172)    Data  0.236 ( 0.137)    Loss 2.6882e-03 (2.8647e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][210/391]    Time  0.041 ( 0.172)    Data  0.000 ( 0.137)    Loss 2.8928e-03 (2.8663e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][220/391]    Time  0.357 ( 0.172)    Data  0.330 ( 0.137)    Loss 2.6964e-03 (2.8684e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][230/391]    Time  0.042 ( 0.171)    Data  0.000 ( 0.136)    Loss 2.5581e-03 (2.8698e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][240/391]    Time  0.322 ( 0.171)    Data  0.295 ( 0.137)    Loss 2.6622e-03 (2.8719e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][250/391]    Time  0.040 ( 0.171)    Data  0.000 ( 0.136)    Loss 2.6410e-03 (2.8833e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][260/391]    Time  0.357 ( 0.171)    Data  0.329 ( 0.136)    Loss 2.4809e-03 (2.8688e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][270/391]    Time  0.041 ( 0.171)    Data  0.000 ( 0.137)    Loss 2.7665e-03 (2.8699e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][280/391]    Time  0.433 ( 0.171)    Data  0.404 ( 0.137)    Loss 2.5914e-03 (2.8610e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][290/391]    Time  0.041 ( 0.171)    Data  0.000 ( 0.136)    Loss 2.4351e-03 (2.8591e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][300/391]    Time  0.365 ( 0.171)    Data  0.337 ( 0.137)    Loss 2.6273e-03 (2.8493e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][310/391]    Time  0.041 ( 0.172)    Data  0.000 ( 0.137)    Loss 4.8451e-03 (2.8486e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][320/391]    Time  0.548 ( 0.172)    Data  0.520 ( 0.138)    Loss 2.6195e-03 (2.8586e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][330/391]    Time  0.042 ( 0.172)    Data  0.000 ( 0.138)    Loss 2.5894e-03 (2.8629e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][340/391]    Time  0.390 ( 0.172)    Data  0.362 ( 0.138)    Loss 2.2831e-03 (2.8722e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][350/391]    Time  0.041 ( 0.173)    Data  0.000 ( 0.138)    Loss 2.4506e-03 (2.8790e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][360/391]    Time  0.332 ( 0.173)    Data  0.304 ( 0.139)    Loss 2.5716e-03 (2.8803e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][370/391]    Time  0.040 ( 0.173)    Data  0.000 ( 0.138)    Loss 5.2257e-03 (2.8855e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][380/391]    Time  0.377 ( 0.173)    Data  0.343 ( 0.139)    Loss 2.4514e-03 (2.8864e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Epoch: [89][390/391]    Time  0.037 ( 0.173)    Data  0.000 ( 0.139)    Loss 2.8977e-03 (2.8924e-03)    Acc@1 100.00 ( 99.98)    Acc@5 100.00 (100.00)
Test: [ 0/40]   Time  0.404 ( 0.404)    Loss 2.7293e+00 (2.7293e+00)    Acc@1  41.02 ( 41.02)   Acc@5  71.48 ( 71.48)
Test: [10/40]   Time  0.015 ( 0.074)    Loss 3.2318e+00 (3.2042e+00)    Acc@1  37.11 ( 36.61)   Acc@5  65.23 ( 64.13)
Test: [20/40]   Time  0.128 ( 0.063)    Loss 3.5614e+00 (3.4057e+00)    Acc@1  32.81 ( 34.56)   Acc@5  55.86 ( 60.53)
Test: [30/40]   Time  0.015 ( 0.057)    Loss 3.7106e+00 (3.4797e+00)    Acc@1  29.30 ( 34.12)   Acc@5  52.73 ( 58.97)
 *   Acc@1 35.090 Acc@5 59.920
Training Finished
```

二、 复现 Word-level Language Model 并讨论

（1）复现训练和文本生成的过程。要求使用 Transformer 模型。提供实验截图

```
| epoch   1 |   200/ 2983 batches | lr 5.00 | ms/batch 13.03 | loss  7.58 | ppl  1956.83
| epoch   1 |   400/ 2983 batches | lr 5.00 | ms/batch 11.91 | loss  6.79 | ppl   891.15
| epoch   1 |   600/ 2983 batches | lr 5.00 | ms/batch 11.90 | loss  6.49 | ppl   660.43
| epoch   1 |   800/ 2983 batches | lr 5.00 | ms/batch 11.94 | loss  6.35 | ppl   574.80
| epoch   1 |  1000/ 2983 batches | lr 5.00 | ms/batch 12.57 | loss  6.25 | ppl   519.56
| epoch   1 |  1200/ 2983 batches | lr 5.00 | ms/batch 11.96 | loss  6.22 | ppl   504.81
| epoch   1 |  1400/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  6.14 | ppl   465.38
| epoch   1 |  1600/ 2983 batches | lr 5.00 | ms/batch 12.11 | loss  6.15 | ppl   470.76
| epoch   1 |  1800/ 2983 batches | lr 5.00 | ms/batch 12.05 | loss  6.03 | ppl   415.32
| epoch   1 |  2000/ 2983 batches | lr 5.00 | ms/batch 12.01 | loss  6.02 | ppl   412.79
| epoch   1 |  2200/ 2983 batches | lr 5.00 | ms/batch 11.99 | loss  5.92 | ppl   373.97
| epoch   1 |  2400/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.94 | ppl   378.05
| epoch   1 |  2600/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.93 | ppl   375.87
| epoch   1 |  2800/ 2983 batches | lr 5.00 | ms/batch 11.99 | loss  5.84 | ppl   343.72
-----------------------------------------------------------------------------------------
| end of epoch   1 | time: 37.67s | valid loss  5.74 | valid ppl   312.56
-----------------------------------------------------------------------------------------
| epoch   2 |   200/ 2983 batches | lr 5.00 | ms/batch 12.43 | loss  5.80 | ppl   329.07
| epoch   2 |   400/ 2983 batches | lr 5.00 | ms/batch 12.11 | loss  5.76 | ppl   318.44
| epoch   2 |   600/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.62 | ppl   275.87
| epoch   2 |   800/ 2983 batches | lr 5.00 | ms/batch 12.03 | loss  5.62 | ppl   277.15
| epoch   2 |  1000/ 2983 batches | lr 5.00 | ms/batch 12.16 | loss  5.60 | ppl   271.13
| epoch   2 |  1200/ 2983 batches | lr 5.00 | ms/batch 12.07 | loss  5.61 | ppl   274.13
| epoch   2 |  1400/ 2983 batches | lr 5.00 | ms/batch 11.96 | loss  5.62 | ppl   274.79
| epoch   2 |  1600/ 2983 batches | lr 5.00 | ms/batch 11.97 | loss  5.66 | ppl   286.68
| epoch   2 |  1800/ 2983 batches | lr 5.00 | ms/batch 11.97 | loss  5.54 | ppl   255.65
| epoch   2 |  2000/ 2983 batches | lr 5.00 | ms/batch 11.97 | loss  5.58 | ppl   264.80
| epoch   2 |  2200/ 2983 batches | lr 5.00 | ms/batch 11.98 | loss  5.48 | ppl   239.90
| epoch   2 |  2400/ 2983 batches | lr 5.00 | ms/batch 11.99 | loss  5.52 | ppl   249.24
| epoch   2 |  2600/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.53 | ppl   251.04
| epoch   2 |  2800/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.45 | ppl   233.63
-----------------------------------------------------------------------------------------
| end of epoch   2 | time: 37.57s | valid loss  5.54 | valid ppl   255.67
-----------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------
| end of epoch   4 | time: 38.32s | valid loss  5.41 | valid ppl   222.56
-----------------------------------------------------------------------------------------
| epoch   5 |   200/ 2983 batches | lr 5.00 | ms/batch 12.00 | loss  5.08 | ppl   161.18
| epoch   5 |   400/ 2983 batches | lr 5.00 | ms/batch 11.98 | loss  5.09 | ppl   163.10
| epoch   5 |   600/ 2983 batches | lr 5.00 | ms/batch 12.04 | loss  4.92 | ppl   136.78
| epoch   5 |   800/ 2983 batches | lr 5.00 | ms/batch 12.06 | loss  4.97 | ppl   143.71
| epoch   5 |  1000/ 2983 batches | lr 5.00 | ms/batch 12.10 | loss  4.97 | ppl   143.40
| epoch   5 |  1200/ 2983 batches | lr 5.00 | ms/batch 12.16 | loss  4.99 | ppl   146.84
| epoch   5 |  1400/ 2983 batches | lr 5.00 | ms/batch 12.11 | loss  5.03 | ppl   153.10
| epoch   5 |  1600/ 2983 batches | lr 5.00 | ms/batch 12.09 | loss  5.08 | ppl   160.80
| epoch   5 |  1800/ 2983 batches | lr 5.00 | ms/batch 12.08 | loss  4.98 | ppl   145.04
| epoch   5 |  2000/ 2983 batches | lr 5.00 | ms/batch 12.07 | loss  5.02 | ppl   151.37
| epoch   5 |  2200/ 2983 batches | lr 5.00 | ms/batch 12.10 | loss  4.90 | ppl   134.87
| epoch   5 |  2400/ 2983 batches | lr 5.00 | ms/batch 12.06 | loss  4.96 | ppl   142.50
| epoch   5 |  2600/ 2983 batches | lr 5.00 | ms/batch 12.05 | loss  4.98 | ppl   145.54
| epoch   5 |  2800/ 2983 batches | lr 5.00 | ms/batch 12.07 | loss  4.92 | ppl   136.58
-----------------------------------------------------------------------------------------
| end of epoch   5 | time: 37.58s | valid loss  5.38 | valid ppl   216.53
-----------------------------------------------------------------------------------------
| epoch   6 |   200/ 2983 batches | lr 5.00 | ms/batch 12.42 | loss  4.95 | ppl   141.23
| epoch   6 |   400/ 2983 batches | lr 5.00 | ms/batch 12.02 | loss  4.97 | ppl   143.79
| epoch   6 |   600/ 2983 batches | lr 5.00 | ms/batch 12.11 | loss  4.79 | ppl   120.69
| epoch   6 |   800/ 2983 batches | lr 5.00 | ms/batch 12.09 | loss  4.85 | ppl   127.36
| epoch   6 |  1000/ 2983 batches | lr 5.00 | ms/batch 12.02 | loss  4.85 | ppl   127.34
| epoch   6 |  1200/ 2983 batches | lr 5.00 | ms/batch 12.04 | loss  4.87 | ppl   129.95
| epoch   6 |  1400/ 2983 batches | lr 5.00 | ms/batch 12.07 | loss  4.92 | ppl   136.44
| epoch   6 |  1600/ 2983 batches | lr 5.00 | ms/batch 12.08 | loss  4.96 | ppl   143.11
| epoch   6 |  1800/ 2983 batches | lr 5.00 | ms/batch 12.06 | loss  4.86 | ppl   129.65
| epoch   6 |  2000/ 2983 batches | lr 5.00 | ms/batch 12.05 | loss  4.90 | ppl   134.91
| epoch   6 |  2200/ 2983 batches | lr 5.00 | ms/batch 12.10 | loss  4.79 | ppl   120.13
| epoch   6 |  2400/ 2983 batches | lr 5.00 | ms/batch 12.03 | loss  4.84 | ppl   126.71
| epoch   6 |  2600/ 2983 batches | lr 5.00 | ms/batch 12.09 | loss  4.87 | ppl   130.33
| epoch   6 |  2800/ 2983 batches | lr 5.00 | ms/batch 12.06 | loss  4.81 | ppl   122.12
-----------------------------------------------------------------------------------------
| end of epoch   6 | time: 37.64s | valid loss  5.37 | valid ppl   214.04
-----------------------------------------------------------------------------------------
=========================================================================================
| End of training | test loss  5.28 | test ppl   195.58
=========================================================================================
```

(2) Transformer 和 CNN 在捕捉上下文依赖上有什么差异？

神经网络结构：Transformer 是基于自注意力机制的序列到序列模型，其中包含 Encoder 和 Decoder 两部分，每个部分包含多个层，相邻层之间使用残差连接和 Layer Normalization 进行连接，利用自注意力机制来获取输入序列的全局信息。而 CNN 则是一种卷积神经网络，其主要特点是使用卷积操作来对输入进行特征提取，对于输入数据的每个不同区域使用不同的权重进行加权汇聚，从而捕捉空间上的局部相关性。

应用场景：Transformer 主要用于序列到序列模型训练任务，例如机器翻译、语音识别等。而 CNN 主要用于计算机视觉任务，如图像分类、物体检测等任务。

模型复杂度：Transformer 由于需要考虑全局信息，因此模型参数较多。CNN 则只需要考虑空间相邻区域的信息，因此模型参数较少。

捕获依赖方式：Transformer 通过自注意力机制来学习序列中不同位置之间的依赖。对于每个位置来说，通过乘以一个权重矩阵并加和得到新的特征向量表示。这种方式可以捕获不同位置之间的长距离依赖。而 CNN 主要是在局部区域内进行卷积操作，对于不同的局部区域使用不同的卷积核进行加权汇聚，从而捕捉局部相关性。