自然语言处理

使用Transformer构建语言模型

- 什么是语言模型：以一个符合语言规律的序列作为输入，模型将利用序列间关系等特征，输出一个在所有词汇上的概率分布，这样的模型称为语言模型。

```
# 语言模型的训练语料一般来自于文章，对应的源文本和目标文本形如：
src1 = "I can do" tgt1 = "can do it"
src2 = "can do it", tgt2 = "do it <eos>"
```

- 语言模型能解决的问题：
    - 根据语言模型的定义，可以在它的基础上完成机器翻译、文本生成等任务，因为我们通过最后输出的概率分布来预测下一个词汇是什么。
    - 语言模型可以判断输入的序列是否为完整的一句话，因为我们可以根据输出的概率分布查看最大概率是否落在句子结束符上，来判断完整性。
    - 语言模型本身的训练目标是预测下一个词，因为它的特征提取部分会抽象很多语言序列之间的关系，这些关系可能同样会对其他语言类任务有效果，因此可以作为预训练模型进行迁移学习。
- 本案例的实现可以分为五个步骤：
    - 导入必备工具包
    - 导入数据集并作基本处理
    - 构建用于模型输入的批次化数据
    - 构建训练和评估函数
    - 进行训练和评估（包括验证及测试）

导入必备的工具包

```
# 数学计算工具包
import math

# torch相关
import torch
import torch.nn as nn
import torch.nn.functional as F

# torch中经典文本数据集有关的工具包
import torchtext

# torchtext中数据处理工具，该函数用于英文分词
from torchtext.data.utils import get_tokenizer

# 已经构建完成的TransformerModel
from pyitcast.transformer import TransformerModel
```

导入wikiText-2数据集并作基本处理

```
# 创建语料域，语料域是存放语料的数据结构，
# 它的四个参数代表给存放语料（或称作文本）施加的作用.
# 分别为 tokenize,使用get_tokenizer("basic_english")获得一个分割器对象，
# 分割方式按照文本为基础英文进行分割.
# init_token为给文本施加的起始符 <sos>给文本施加的终止符<eos>，
# 最后一个lower为True，存放的文本字母全部小写.
TEXT = torchtext.data.Field(tokenize=get_tokenizer("basic_english"),
                            init_token='<sos>',
                            eos_token='<eos>',
                            lower=True)

# 然后使用torchtext的数据集方法导入数据
# 并切分为训练文本，验证文本，测试文本，并对这些文本施加刚刚创建的语料域
train_txt, val_txt, test_txt = torchtext.datasets.WikiText2.splits(TEXT)


# 将训练集文本数据构建一个vocab对象
# 可以用vocab对象的stoi方法统计文本共包含的不重复词汇总数
TEXT.build_vocab(train_txt)

# 然后选择设备
device = torch.device("cuda")
```

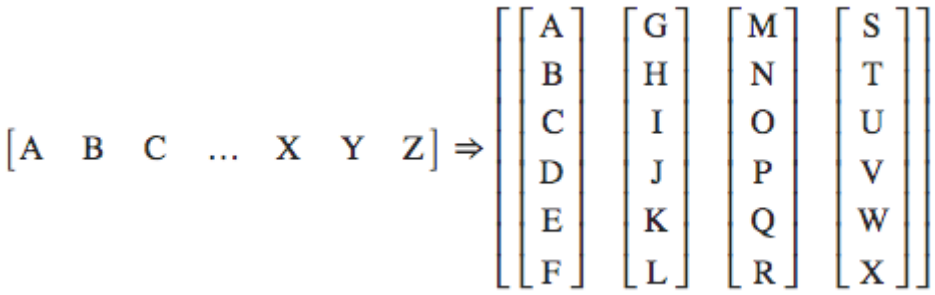
构建用于模型输入的批次化数据

- 批次化过程的第一个函数batchify代码分析

```
def batchify(data, bsz):
    """
    该函数用于将文本数据映射成连续数字，并转换指定的样式，指定的样式可参考图片
    :param data: 之前得到的文本数据(train_txt, val_txt, test_txt)
    :param bsz: batch_size，每次模型更新参数的数据量
    :return: 处理之后的数据
    """
    # 先将单词映射成连续对应的数字
    data = TEXT.numericalize([data.examples[0].text])

    # 接着用数据词汇总数除bsz并取整得到一个nbatch代表需要多少次batch后遍历所有数据
    nbatch = data.size(0) // bsz

    # 使用narrow方法对不规整剩余数据进行删除
    # 第一个参数代表横轴删除还是纵轴删除，0为横，1为纵
    # 第二个和第三个参数代表保留开始轴到结束轴的数值，类似于切片
    data = data.narrow(0, 0, nbatch*bsz)

    data = data.view(bsz, -1).t().contiguous()
    return data.to(device)


# 用batchify来处理训练数据，验证数据以及测试数据
# 训练数据的bsz
batch_size = 20

# 验证和测试数据（统称为评估数据）的bsz
```

```
eval_batch_size = 10

# 获得处理后的数据
train_data = batchify(train_txt, batch_size)
val_data = batchify(val_txt, eval_batch_size)
test_data = batchify(test_txt, eval_batch_size)
```

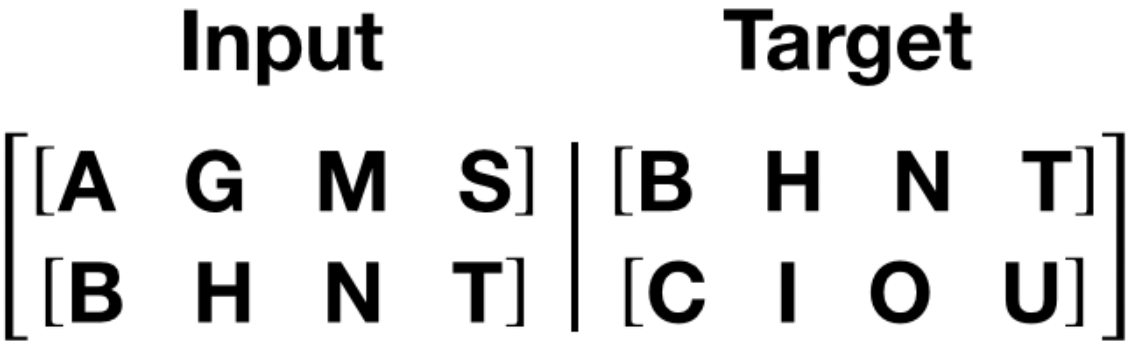
- batchify的样式转化图，其中每个字母代表句子中的一个单词：



- torch.narrow演示：

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> x.narrow(0, 0, 2)
tensor([[ 1,  2,  3],
        [ 4,  5,  6]])
>>> x.narrow(1, 1, 2)
tensor([[ 2,  3],
        [ 5,  6],
        [ 8,  9]])
```

- 上面的分割批次并没有进行源数据与目标数据的处理，接下来我们将根据语言模型训练的语料规定来构建源数据与目标数据
- 语言模型训练的语料规定：
  - 如果源数据句子为ABCD，则目标数据为BCDE



- 图中句子序列是竖着的，发现如果用一个批次处理所有数据，以训练数据为例，每个句子长度高达104335，这明显是不科学的。因此我们要限定每个批次中句子长度允许的最大值bptt
- 下面是批次化过程第二个函数代码分析：

```
# 设置句子最大长度35
bptt = 35

def get_batch(source, i):
```

```
    """
    用于获取每个批次合理大小的源数据和目标数据
    :param source: 通过batchify得到的三个data
    :param i: 具体批次次数
    :return: 源数据与目标数据
    """
    # 确定句子长度，应该是bptt和len(source)-1-i的小值
    seq_len = min(bptt, len(source)-1-i)

    # 语言模型训练的源数据的第i批次数据将是batchify结果切片
    data = source[i:i+seq_len]

    # 根据语言模型训练的语料规定，他的目标数据是源数据后移一位
    # 最后目标数据的切片会越界，所以使用view(-1)保证形状正常
    target = source[i+1:i+1+seq_len].view(-1)
    return data, target
```

- 输入示例:

```
get_batch(test_data, 1)
Out[3]:
(tensor([[   12,  1053,   355,   134,    37,     7,     4,     0,   835,  9834],
         [  635,     8,     5,     5,   421,     4,    88,     8,   573,  2511],
         [    0,    58,     8,     8,     6,   692,   544,     0,   212,     5],
         [   12,     0,   105,    26,     3,     5,     6,     0,     4,    56],
         [    3, 16074, 21254,   320,     3,   262,    16,     6,  1087,    89],
         [    3,   751,  3866,    10,    12,    31,   246,   238,    79,    49],
         [  635,   943,    78,    36,    12,   475,    66,    10,     4,   924],
         [    0,  2358,    52,     4,    12,     4,     5,     0, 19831,    21],
         [   26,    38,    54,    40,  1589,  3729,  1014,     5,     8,     4],
         [   33, 17597,    33,  1661,    15,     7,     5,     0,     4,   170],
         [  335,   268,   117,     0,     0,     4,  3144,  1557,     0,   160],
         [  106,     4,  4706,  2245,    12,  1074,    13,  2105,     5,    29],
         [    5, 16074,    10,  1087,    12,   137,   251, 13238,     8,     4],
         [  394,   746,     4,     9,    12,  6032,     4,  2190,   303, 12651],
         [    8,   616,  2107,     4,     3,     4,   425,     0,    10,   510],
         [ 1339,   112,    23,   335,     3, 22251,  1162,     9,    11,     9],
         [ 1212,   468,     6,   820,     9,     7,  1231,  4202,  2866,   382],
         [    6,    24,   104,     6,     4,     4,     7,    10,     9,   588],
         [   31,   190,     0,     0,   230,   267,     4,   273,   278,     6],
         [   34,    25,    47,    26,  1864,     6,   694,     0,  2112,     3],
         [   11,     6,    52,   798,     8,    69,    20,    31,    63,     9],
         [ 1800,    25,  2141,  2442,   117,    31,   196,  7290,     4,   298],
         [   15,   171,    15,    17,  1712,    13,   217,    59,   736,     5],
         [ 4210,   191,   142,    14,  5251,   939,    59,    38, 10055, 25132],
         [  302,    23, 11718,    11,    11,   599,   382,   317,     8,    13],
         [   16,  1564,     9,  4808,     6,     0,     6,     6,     4,     4],
         [    4,     7,    39,     7,  3934,     5,     9,     3,  8047,   557],
         [  394,     0, 10715,  3580,  8682,    31,   242,     0, 10055,   170],
         [   96,     6,   144,  3403,     4,    13,  1014,    14,     6,  2395],
         [    4,     3, 13729,    14,    40,     0,     5,    18,   676,  3267],
         [ 1031,     3,     0,   628,  1589,    22, 10916, 10969,     5, 22548],
         [    9,    12,     6,    84,    15,    49,  3144,     7,   102,    15],
         [  916,    12,     4,   203,     0,   273,   303,   333,  4318,     0],
         [    6,    12,     0,  4842,     5,    17,     4,    47,  4138,  2072],
         [   38,   237,     5,    50,    35,    27, 18530,   244,    20,
6]]),
```

```
tensor([  635,     8,     5,     5,   421,     4,    88,     8,   573,  2511,
             0,    58,     8,     8,     6,   692,   544,     0,   212,     5,
            12,     0,   105,    26,     3,     5,     6,     0,     4,    56,
             3, 16074, 21254,   320,     3,   262,    16,     6,  1087,    89,
             3,   751,  3866,    10,    12,    31,   246,   238,    79,    49,
           635,   943,    78,    36,    12,   475,    66,    10,     4,   924,
             0,  2358,    52,     4,    12,     4,     5,     0, 19831,    21,
            26,    38,    54,    40,  1589,  3729,  1014,     5,     8,     4,
            33, 17597,    33,  1661,    15,     7,     5,     0,     4,   170,
           335,   268,   117,     0,     0,     4,  3144,  1557,     0,   160,
           106,     4,  4706,  2245,    12,  1074,    13,  2105,     5,    29,
             5, 16074,    10,  1087,    12,   137,   251, 13238,     8,     4,
           394,   746,     4,     9,    12,  6032,     4,  2190,   303, 12651,
             8,   616,  2107,     4,     3,     4,   425,     0,    10,   510,
          1339,   112,    23,   335,     3, 22251,  1162,     9,    11,     9,
          1212,   468,     6,   820,     9,     7,  1231,  4202,  2866,   382,
             6,    24,   104,     6,     4,     4,     7,    10,     9,   588,
            31,   190,     0,     0,   230,   267,     4,   273,   278,     6,
            34,    25,    47,    26,  1864,     6,   694,     0,  2112,     3,
            11,     6,    52,   798,     8,    69,    20,    31,    63,     9,
          1800,    25,  2141,  2442,   117,    31,   196,  7290,     4,   298,
            15,   171,    15,    17,  1712,    13,   217,    59,   736,     5,
          4210,   191,   142,    14,  5251,   939,    59,    38, 10055, 25132,
           302,    23, 11718,    11,    11,   599,   382,   317,     8,    13,
            16,  1564,     9,  4808,     6,     0,     6,     6,     4,     4,
             4,     7,    39,     7,  3934,     5,     9,     3,  8047,   557,
           394,     0, 10715,  3580,  8682,    31,   242,     0, 10055,   170,
            96,     6,   144,  3403,     4,    13,  1014,    14,     6,  2395,
             4,     3, 13729,    14,    40,     0,     5,    18,   676,  3267,
          1031,     3,     0,   628,  1589,    22, 10916, 10969,     5, 22548,
             9,    12,     6,    84,    15,    49,  3144,     7,   102,    15,
           916,    12,     4,   203,     0,   273,   303,   333,  4318,     0,
             6,    12,     0,  4842,     5,    17,     4,    47,  4138,  2072,
            38,   237,     5,    50,    35,    27, 18530,   244,    20,     6,
            13,  1083,    35,  1990,   653,    13,    10,    11,  1538,    56]))
```

构建训练和评估函数

- 设置模型超参数和初始化模型

```
# 通过TEXT.vocab.stoi属性获得不重复词汇总数
ntokens = len(TEXT.vocab.stoi)
# 词嵌入大小
emsize = 200
# 前馈全连接层节点数
nhid = 200
# 编码器层数量
nlayers = 2
# 多头注意力机制头数
nhead = 2
# 置0比率
dropout = 0.2

# 将参数输入到模型中
```

```
model = TransformerModel(ntokens, emsize, nhead, nhid, nlayers,
dropout).to(device)

# 模型初始化后，接下来进行损失函数和优化方法的选择
# 使用nn自带的交叉熵损失
criterion = nn.CrossEntropyLoss()

# 初始学习率
lr = 5.0

# 优化器选择torch自带的SGD随机梯度下降方法
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

# 学习率调整方法，使用torch自带的lr_scheduler，将优化器传入
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.95)
```

- 模型训练代码分析:

```
def train(epoch):
    """
    训练函数
    :param epoch:循环次数
    :return: None
    """
    # 模型开启训练模式
    model.train()
    total_loss = 0.
    start_time = time.time()
    plot_losses = []
    # 遍历批次数据
    for batch, i in enumerate(range(0, train_data.size(0) - 1, bptt)):
        # 获取源数据和目标数据
        data, targets = get_batch(train_data, i)
        # 设置初始梯度为0
        optimizer.zero_grad()
        # 装入model得到输出
        output = model(data)
        # 将输出和目标数据传入损失函数对象
        loss = criterion(output.view(-1, ntokens), targets)
        # 反向传播获得总损失
        loss.backward()
        # 使用nn自带的clip_grad_norm_方法进行梯度规范化，防止出现梯度爆炸或消失
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5)
        # 更新模型参数
        optimizer.step()
        # 损失加和
        total_loss += loss.item()
        # 日志打印间隔
        log_interval = 200
        # 如果batch是200的倍数，则打印日志
        if batch % log_interval == 0 and batch > 0:
            # 平均损失
            cur_loss = total_loss / log_interval
            # 需要的时间
            elapsed = time.time() - start_time
            # 打印轮数、当前批次和总批次，当前学习率，训练速度
            # 平均损失，以及困惑度
```

```
            # 困惑度是衡量语言模型的重要指标，他是交叉熵平均损失取自然对数的底数
            print('| epoch {:3d} | {:5d}/{:5d} batches | '
                  'lr {:02.2f} | ms/batch {:5.2f} | '
                  'loss {:5.2f} | ppl {:8.2f}'.format(
                epoch, batch, len(train_data) // bptt, scheduler.get_lr()[0],
                elapsed * 1000 / log_interval,
                cur_loss, math.exp(cur_loss)
            ))
        # 作图间隔
        plt_interval = 50
        plot_loss_total += loss
        # 做出损失曲线的图
        if batch % plt_interval == 0 and batch > 0:
            plot_loss_avg = plot_loss_total / plt_interval
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0

        plt.figure()
        plt.plot(plot_losses)
        plt.savefig('./learn_loss.png')
        # 每个批次结束后，总损失归0
        total_loss = 0
        # 开始时间取当前时间
        start_time = time.time()
```

- 模型评估代码分析：

```
def evaluate(eval_model, data_source):
    """
    评估函数
    :param eval_model:每轮训练产生的模型
    :param data_source: 验证或测试数据集
    :return: 平均损失
    """
    # 模型开启评估模式
    eval_model.eval()
    # 损失归零
    total_loss = 0
    # 因为评估模式模型参数不变，所以不进行反向传播
    with torch.no_grad():
        for i in range(0, data_source(0)-1, bptt):
            data, targets = get_batch(data_source, i)
            output = eval_model(data)
            output_flat = output.view(-1, ntokens)
            total_loss += criterion(output_flat, targets).item()

            cur_loss = total_loss / ((data_source.size(0)-1) / bptt)
    return cur_loss
```

进行训练和评估（包括验证和测试）

- 模型训练与验证代码分析：

```
# 初始化最佳验证损失，初始值无穷大
import copy
best_val_loss = float('inf')
```

```
# 训练轮数
epochs = 3

# 定义最佳模型变量，初值为None
best_model = None

if __name__ == '__main__':
    for epoch in range(1, epochs + 1):
        # 获得轮数开始时间
        epoch_start_time = time.time()
        # 调用训练函数
        train(epoch)
        # 训练后模型参数发生了变化
        # 将模型和评估数据传入评估函数中
        val_loss = evaluate(model, val_data)
        # 打印每轮的评估日志
        print('-'*50)
        print('| end of epoch {:3d} | time: {:5.2f}s | valid loss {:5.2f} |
valid ppl {:8.2f}'.format(
            epoch, (time.time() - epoch_start_time), val_loss,
math.exp(val_loss)
        ))
        print('-'*50)

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_model = copy.deepcopy(model)
        # 每轮都会对优化方法的学习率进行调整
        scheduler.step()
```