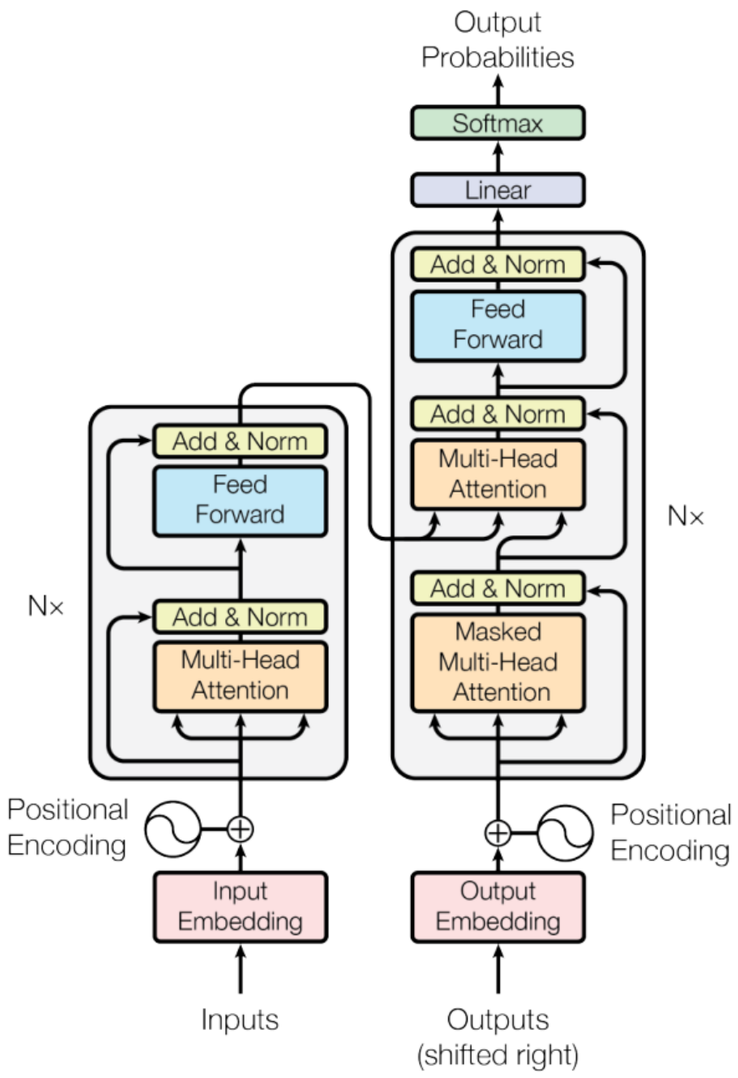
第7章 Transformer

参考链接：
https://www.zhihu.com/question/362131975
https://zhuanlan.zhihu.com/p/54675834
http://nlp.seas.harvard.edu/2018/04/03/attention.html
实现细节建议：用草稿纸，画出每个小组件，以及小组件拼装成大组件



In [1]:
```
import os,sys,abc,copy,math
import torch,nltk,jieba
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import defaultdict,Counter
torch.manual_seed(102)
```

Out[1]:
```
<torch._C.Generator at 0x276cbb19cb0>
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

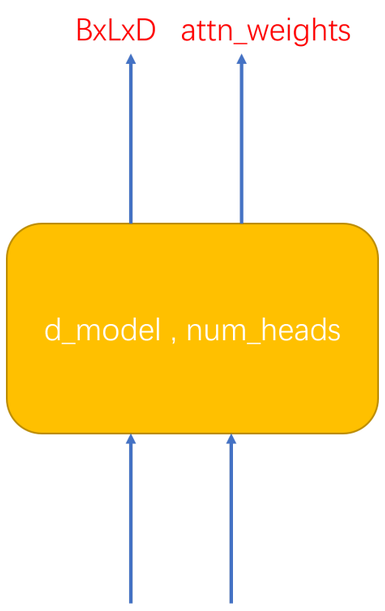
In [3]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device:', device)
torch.set_default_tensor_type('torch.cuda.FloatTensor')
```

```
Device: cuda
```

第1节 三个小组件

组件的初始化参数写在里面，输出输出用箭头指出

多头自注意力层



In [4]:
```
class MultiheadAttention(nn.Module):
    def __init__(self,d_model,num_heads):
        super(MultiheadAttention,self).__init__()

        assert d_model % num_heads==0
        self.d_model=d_model
        self.num_heads=num_heads
        self.head_size=torch.tensor(d_model/num_heads).int()

        self.W_q=nn.Linear(d_model,d_model,bias=False)
        self.W_k=nn.Linear(d_model,d_model,bias=False)
        self.W_v=nn.Linear(d_model,d_model,bias=False)
        self.W_out=nn.Linear(d_model,d_model,bias=False)
        self.softmax=nn.Softmax(dim=-1)

    def forward(self,q,k,v,key_padding_mask=None,attn_mask=None,average_attn_weights=True):
        B,L1,D=q.shape
        L2=k.shape[1]
        Q,K,V=self.W_q(q),self.W_k(k),self.W_v(v)                                   # Q=[B,L1,D], K,V=[B,L2,D]
        Q,K,V=[self.split_head_reshape(item,self.num_heads,self.head_size) for item in [Q,K,V]]   # Q=[B*num_heads,L1,head_size] , K,V=[B*num_heads,L2,head_size]
        attn_score=torch.matmul(Q,K.transpose(1,2))/torch.sqrt(self.head_size)     # [B*num_heads,L1,L2]

        # 分别掩码，官方代码是合并到attn_mask再掩码
        if key_padding_mask is not None:
            assert key_padding_mask.shape==torch.Size([B,L2])
            key_padding_mask = key_padding_mask.reshape(B, 1, 1, L2).expand([-1, self.num_heads, -1, -1]).reshape([B * self.num_heads, 1, L2])
            attn_score=attn_score.masked_fill(key_padding_mask,-1e9)
        if attn_mask is not None:
            if attn_mask.dim()==2:
                assert attn_mask.shape==torch.Size([L2,L2])
```

```
            attn_score=attn_score.masked_fill(attn_mask,-1e9)
        elif attn_mask.dim()==3:
            assert attn_mask.shape==torch.Size([B*self.num_heads,L2,L2])
            attn_score=attn_score.masked_fill(attn_mask,-1e9)

    attention=self.softmax(attn_score)             # [B*num_heads,L1,L2]
    out=torch.matmul(attention,V)                   # [B*num_heads,L1,head_size]
    out=self.head_rebuild(out,B,L1,self.num_heads,self.head_size)
    out=self.W_out(out)

    attn_weights=attention.reshape([B,self.num_heads,L1,L2])
    if average_attn_weights:
        attn_weights=attn_weights.mean(dim=1)

    return out,attn_weights

def split_head_reshape(self,X,num_heads,head_size):
    B,L,D=X.shape
    X=X.reshape([B,L,num_heads,head_size])
    X=X.transpose(1,2)
    X=X.reshape([B*num_heads,L,head_size])
    return X
def head_rebuild(self,X,B,L,num_heads,head_size):
    X=X.reshape([B,num_heads,L,head_size])
    X=X.transpose(1,2)
    X=X.reshape([B,L,num_heads*head_size])
    return X
```

In [130…

```
def creat_key_padding_mask(L,seq_lens):
    if type(seq_lens) == list:
        seq_lens=torch.tensor(seq_lens)
    seq_lens=seq_lens.unsqueeze(-1)  # Bx1
    mask=torch.arange(L)<seq_lens    # BxL
    return ~mask
def creat_attn_mask(L):
    if L==1:
        return None
    mask=torch.arange(L).unsqueeze(-1)<torch.arange(L).unsqueeze(0)
    return mask
```

In [6]:

```
# test
input_x=torch.randn([3,4,6])
seq_lens=torch.tensor([4,3,2])
src_key_padding_mask=creat_key_padding_mask(4,seq_lens)
tgt_attn_mask=creat_attn_mask(4)
src_key_padding_mask,tgt_attn_mask
```

Out[6]:

```
(tensor([[False, False, False, False],
         [False, False, False,  True],
         [False, False,  True,  True]]),
 tensor([[False,  True,  True,  True],
         [False, False,  True,  True],
         [False, False, False,  True],
         [False, False, False, False]]))
```

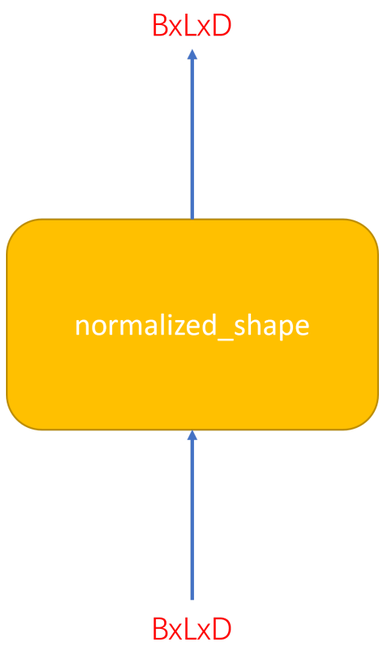
In [7]:

```
multi_head_attention=MultiheadAttention(d_model=6,num_heads=2)
out,attn=multi_head_attention(input_x,input_x,input_x,key_padding_mask=src_key_padding_mask,attn_mask=tgt_attn_mask)
out,attn
```

Out[7]:

```
(tensor([[[ 0.2029,  0.5731,  0.4110, -0.4776,  0.0863,  0.4328],
          [ 0.5019, -0.0799,  0.3818, -0.7143, -0.1312,  0.1487],
          [-0.0284, -0.0241,  0.0591,  0.1632,  0.1279,  0.1031],
          [ 0.3566,  0.1524,  0.2976, -0.6639, -0.1322,  0.0904]],

         [[ 0.0513, -0.6195, -0.0673,  0.3932,  0.1076,  0.1155],
          [-0.2291, -0.0871, -0.0694,  0.3107,  0.0826, -0.0380],
          [-0.1954,  0.1882,  0.0255, -0.0665, -0.0236, -0.0441],
          [-0.1742,  0.0458, -0.0698,  0.1455,  0.0406, -0.0137]],

         [[ 0.3695, -0.2483,  0.3822, -0.5670, -0.1171, -0.0818],
          [-0.0846,  0.1005,  0.1521, -0.1970,  0.0164,  0.0258],
          [-0.0861,  0.0778,  0.1734, -0.1927,  0.0072, -0.0039],
          [-0.0380, -0.0372,  0.2698, -0.2219, -0.0386, -0.1138]]],
        grad_fn=<UnsafeViewBackward0>),
 tensor([[[1.0000, 0.0000, 0.0000, 0.0000],
          [0.3546, 0.6454, 0.0000, 0.0000],
          [0.2964, 0.1419, 0.5617, 0.0000],
          [0.1658, 0.3939, 0.0888, 0.3515]],

         [[1.0000, 0.0000, 0.0000, 0.0000],
          [0.4734, 0.5266, 0.0000, 0.0000],
          [0.2551, 0.2201, 0.5247, 0.0000],
          [0.3600, 0.3670, 0.2731, 0.0000]],

         [[1.0000, 0.0000, 0.0000, 0.0000],
          [0.5654, 0.4346, 0.0000, 0.0000],
          [0.5223, 0.4777, 0.0000, 0.0000],
          [0.4322, 0.5678, 0.0000, 0.0000]]], grad_fn=<MeanBackward1>))
```

加与规范层



In [8]:

```
class LayerNorm(nn.Module):
    def __init__(self, normalized_shape, eps=1e-5):
        super(LayerNorm, self).__init__()
        self.normalized_shape = normalized_shape
        self.eps = eps
        self.gamma =  nn.Parameter(torch.ones(normalized_shape))
        self.beta = nn.Parameter(torch.zeros(normalized_shape))
        nn.init.ones_(self.gamma)
        nn.init.zeros_(self.beta)

    def forward(self, input):
        mean = input.mean(dim=-1, keepdim=True)
        variance = input.var(dim=-1, keepdim=True)
        input = (input - mean) / torch.sqrt(variance + self.eps)
        input = input * self.gamma + self.beta    # 这里是逐元素乘法
        return input
```

In [9]:

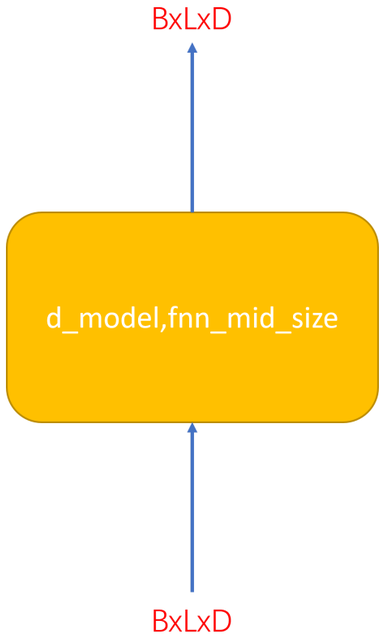
```
class AddNorm(nn.Module):
    def __init__(self, normalized_shape, eps=1e-5):
        super(AddNorm, self).__init__()
        self.layer_norm=LayerNorm(normalized_shape)

    def forward(self,X,H):
        return self.layer_norm(X+H)
```

In [10]:

```
# test
B,L,D=out.shape
addnorm=AddNorm(normalized_shape=D)
out2=addnorm(out,out)
out2
```

Out[10]:
```
tensor([[[-4.9330e-03,  9.7756e-01,  5.4731e-01, -1.8110e+00, -3.1429e-01,
           6.0530e-01],
         [ 1.1096e+00, -2.2402e-01,  8.3424e-01, -1.6781e+00, -3.4164e-01,
           2.9995e-01],
         [-1.1948e+00, -1.1408e+00, -9.6848e-02,  1.2098e+00,  7.6717e-01,
           4.5556e-01],
         [ 9.0569e-01,  3.6142e-01,  7.4847e-01, -1.8146e+00, -3.9713e-01,
           1.9610e-01]],

        [[ 1.6139e-01, -1.8247e+00, -1.8975e-01,  1.1736e+00,  3.2804e-01,
           3.5141e-01],
         [-1.2169e+00, -4.4562e-01, -3.4922e-01,  1.7146e+00,  4.7588e-01,
          -1.7881e-01],
         [-1.4013e+00,  1.6514e+00,  3.5699e-01, -3.7566e-01, -3.4291e-02,
          -1.9710e-01],
         [-1.5481e+00,  4.5667e-01, -5.9677e-01,  1.3648e+00,  4.0932e-01,
          -8.5952e-02]],

        [[ 1.1250e+00, -5.5679e-01,  1.1595e+00, -1.4245e+00, -1.9958e-01,
          -1.0362e-01],
         [-6.8591e-01,  7.7634e-01,  1.1842e+00, -1.5736e+00,  1.1249e-01,
           1.8648e-01],
         [-6.4608e-01,  6.4436e-01,  1.3973e+00, -1.4854e+00,  8.8753e-02,
           1.0139e-03],
         [-4.9100e-02, -4.4503e-02,  1.8315e+00, -1.1728e+00, -5.2893e-02,
          -5.1223e-01]]], grad_fn=<AddBackward0>)
```

全连接前馈层



In [11]:
```
class FNN(nn.Module):
    def __init__(self,d_model,fnn_mid_size):
        super(FNN,self).__init__()
        self.W1=nn.Linear(d_model,fnn_mid_size)
        self.W2=nn.Linear(fnn_mid_size,d_model)
    def forward(self,X):
        return self.W2(F.relu(self.W1(X)))
```

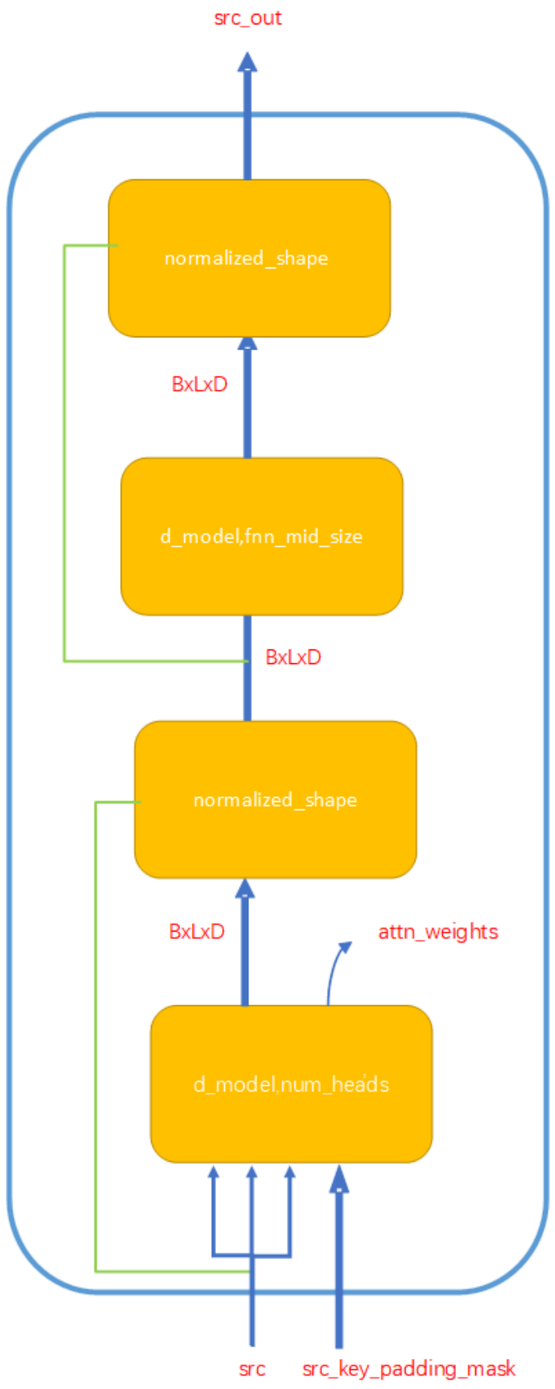
In [12]:
```
# test
fnn=FNN(d_model=D,fnn_mid_size=10)
out3=fnn(out2)
out3
```

Out[12]:
```
tensor([[[ 0.4199, -0.0234, -0.3344,  0.4114, -0.1172,  0.0310],
         [ 0.1540, -0.0219, -0.2856, -0.0436, -0.2803, -0.2986],
         [ 0.2609, -0.0717, -0.6332, -0.0983, -0.1818, -0.3899],
         [ 0.2606,  0.0100, -0.2876,  0.0734, -0.1747, -0.1845]],

        [[ 0.2631,  0.0044, -0.5364, -0.0703, -0.3490, -0.6079],
         [ 0.3715,  0.0449, -0.4952,  0.0267, -0.1213, -0.3447],
         [ 0.4425, -0.0095, -0.4321,  0.4979,  0.0237,  0.0770],
         [ 0.3474,  0.1112, -0.4667,  0.1590, -0.0116, -0.1906]],

        [[ 0.0612, -0.1265, -0.3129, -0.1904, -0.2589, -0.3474],
         [ 0.4004, -0.2126, -0.4828,  0.3230, -0.1012,  0.0064],
         [ 0.3681, -0.2443, -0.5148,  0.2470, -0.0612, -0.0364],
         [ 0.1912, -0.2600, -0.5490, -0.0862,  0.0056, -0.2409]]],
       grad_fn=<ViewBackward0>)
```

第2节 两个Block单元

初始化需要的参数就是里面小组件的初始化参数，然后就是拼接

编码器的Block单元

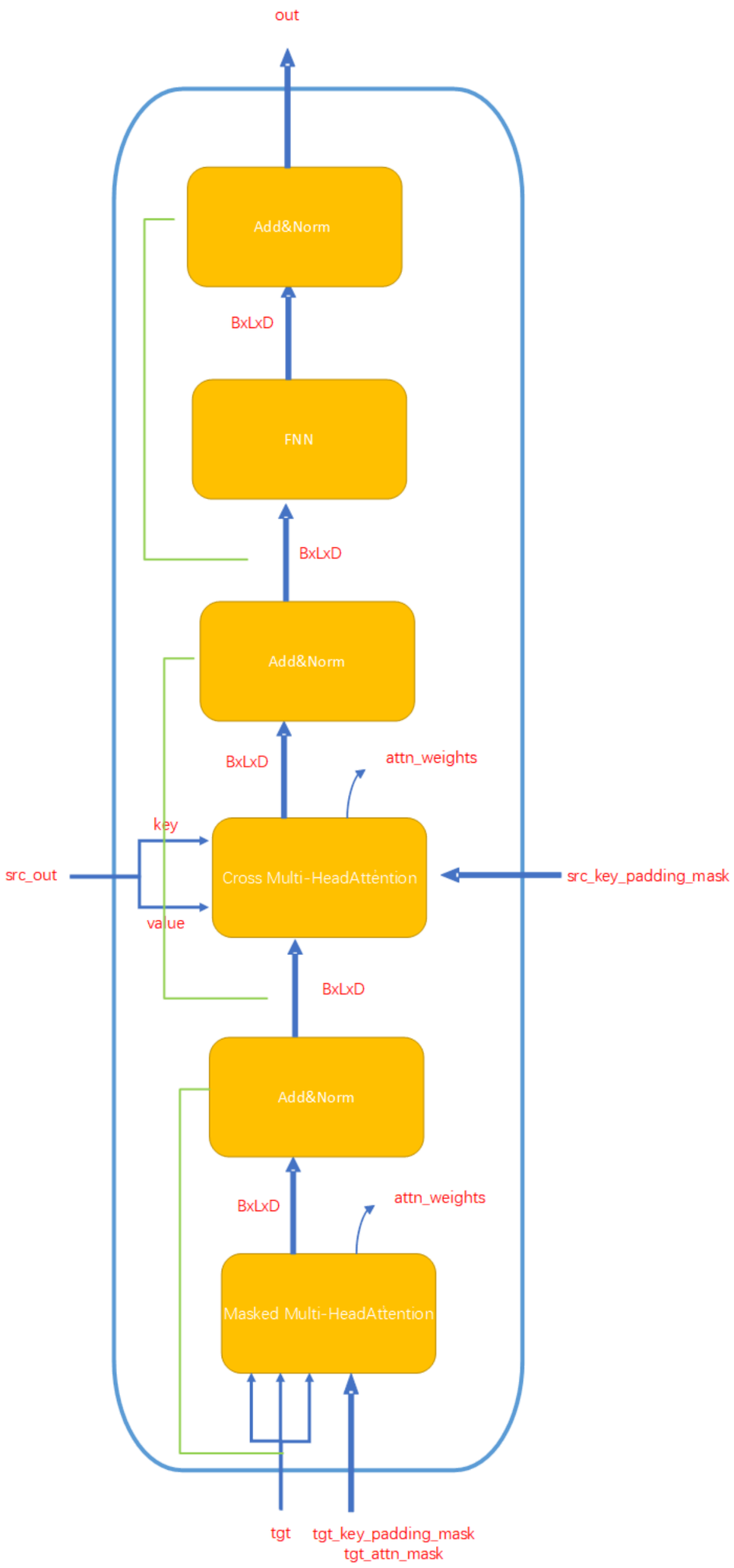


In [13]:
```
class EncoderBlock(nn.Module):
    def __init__(self,d_model=None,num_heads=1,normalized_shape=None,fnn_mid_size=None):
        super(EncoderBlock,self).__init__()
        self.multi_head_attention=MultiheadAttention(d_model,num_heads)
        self.addnorm1=AddNorm(normalized_shape)
        self.fnn=FNN(d_model,fnn_mid_size)
        self.addnorm2=AddNorm(normalized_shape)
        self.attn_weights=None                      # 用于注意力可视化
    def forward(self,X,src_key_padding_mask=None):
        H,attn_weights=self.multi_head_attention(X,X,X,src_key_padding_mask)
        X=self.addnorm1(X,H)
        H=self.fnn(X)
        X=self.addnorm2(X,H)
        self.attn_weights=attn_weights
        return X
```

In [14]:
```
# test
encoder=EncoderBlock(d_model=D,num_heads=2,normalized_shape=D,fnn_mid_size=10)
out4=encoder(input_x,src_key_padding_mask=src_key_padding_mask)
out4,encoder.attn_weights
```

Out[14]:
```
(tensor([[[-0.2300,  1.4451,  0.3565,  0.0376,  0.0425, -1.6517],
          [ 0.9072, -1.7957,  0.0911,  0.5093,  0.7112, -0.4230],
          [-1.2505,  0.8657, -1.2794,  0.3590,  0.4667,  0.8385],
          [ 0.3512, -0.3901,  1.6110, -0.0455, -0.0698, -1.4568]],

         [[-0.7831,  0.7505, -1.1250, -0.7036,  0.5238,  1.3373],
          [-1.8306,  0.6562,  0.8613,  0.1840, -0.4013,  0.5305],
          [-0.0019, -0.7277,  1.8291,  0.0475, -1.0556, -0.0915],
          [-0.1263,  0.5895,  0.3677, -1.5378,  1.3247, -0.6178]],

         [[-0.8983, -0.2419,  0.3160, -0.4241,  1.8642, -0.6159],
          [-0.2183,  0.1836, -0.2230,  1.3434, -1.6552,  0.5694],
          [ 1.2794, -0.6786, -0.9396, -0.0897, -0.7655,  1.1940],
          [-0.5969, -1.1300,  1.7377,  0.1058, -0.4641,  0.3475]]],
        grad_fn=<AddBackward0>),
 tensor([[[0.2278, 0.3204, 0.1592, 0.2926],
          [0.2445, 0.2507, 0.2393, 0.2655],
          [0.2325, 0.2772, 0.1744, 0.3159],
          [0.2553, 0.2554, 0.2908, 0.1985]],

         [[0.2460, 0.3590, 0.3950, 0.0000],
          [0.4048, 0.3096, 0.2856, 0.0000],
          [0.4747, 0.2906, 0.2347, 0.0000],
          [0.2535, 0.3399, 0.4066, 0.0000]],

         [[0.5147, 0.4853, 0.0000, 0.0000],
          [0.3144, 0.6856, 0.0000, 0.0000],
          [0.3753, 0.6247, 0.0000, 0.0000],
          [0.3598, 0.6402, 0.0000, 0.0000]]], grad_fn=<MeanBackward1>))
```

解码器的Block单元



In [15]:
```
class DecoderBlock(nn.Module):
    def __init__(self,d_model=None,num_heads=1,normalized_shape=None,fnn_mid_size=None):
        super(DecoderBlock,self).__init__()
        self.masked_multi_head_attention=MultiheadAttention(d_model,num_heads)
        self.addnorm1=AddNorm(normalized_shape)
        self.cross_multi_head_attention=MultiheadAttention(d_model,num_heads)
        self.addnorm2=AddNorm(normalized_shape)
        self.fnn=FNN(d_model,fnn_mid_size)
        self.addnorm3=AddNorm(normalized_shape)
        self.cross_attn=None  # 这个用于可视化
    def forward(self,tgt=None,memory=None,src_key_padding_mask=None,tgt_key_padding_mask=None,tgt_attn_mask=None):
        H,tgt_attn=self.masked_multi_head_attention(tgt,tgt,tgt,key_padding_mask=tgt_key_padding_mask,attn_mask=tgt_attn_mask)
        X=self.addnorm1(tgt,H)
        H,cross_attn=self.cross_multi_head_attention(X,memory,memory,key_padding_mask=src_key_padding_mask)
        X=self.addnorm2(X,H)
        H=self.fnn(X)
        X=self.addnorm3(X,H)
        self.cross_attn=cross_attn
        return X
```

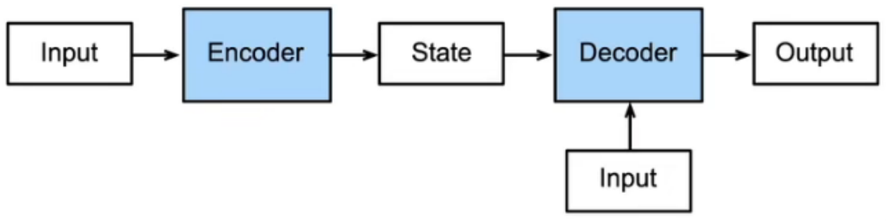
In [16]:
```
# test
decoder=DecoderBlock(d_model=D,num_heads=2,normalized_shape=D,fnn_mid_size=10)
out5=decoder(memory=out4,tgt=input_x,src_key_padding_mask=src_key_padding_mask,tgt_key_padding_mask=src_key_padding_mask,tgt_attn_mask=tgt_attn_mask)
out5,decoder.cross_attn
```

Out[16]:
```
(tensor([[[-0.0042,  1.8419, -0.4406, -0.0972, -0.1212, -1.1787],
          [ 0.4874, -1.6536, -0.1107,  0.7072,  1.1074, -0.5377],
          [-0.7437,  0.7611, -1.6936,  0.4114,  0.5099,  0.7549],
          [ 0.2464, -0.0772,  1.1751,  0.7305, -0.3793, -1.6956]],

         [[-0.1528,  0.4519, -1.2535, -0.9607,  0.4860,  1.4290],
          [-1.7980,  0.8224,  0.3899, -0.2715, -0.0703,  0.9275],
          [-0.1989, -0.3764,  1.7481,  0.2869, -1.2844, -0.1754],
          [ 0.2528,  1.0737, -0.4002, -1.6565,  0.9185, -0.1883]],

         [[-0.7884, -0.0037, -0.1317, -0.3279,  1.9431, -0.6913],
          [-0.3179,  0.1033, -0.3524,  1.7162, -1.3357,  0.1865],
          [ 1.0172, -0.9894, -1.0721,  0.4293, -0.5479,  1.1629],
          [-0.9704, -1.1018,  1.6373,  0.3484, -0.0953,  0.1818]]],
        grad_fn=<AddBackward0>),
 tensor([[[0.2573, 0.2372, 0.2559, 0.2495],
          [0.2178, 0.2872, 0.2062, 0.2887],
          [0.2811, 0.1913, 0.3362, 0.1914],
          [0.1982, 0.3033, 0.1968, 0.3016]],

         [[0.3997, 0.3603, 0.2400, 0.0000],
          [0.3552, 0.3256, 0.3191, 0.0000],
          [0.2317, 0.2906, 0.4777, 0.0000],
          [0.2674, 0.3398, 0.3929, 0.0000]],

         [[0.4925, 0.5075, 0.0000, 0.0000],
          [0.4632, 0.5368, 0.0000, 0.0000],
          [0.6102, 0.3898, 0.0000, 0.0000],
          [0.3032, 0.6968, 0.0000, 0.0000]]], grad_fn=<MeanBackward1>))
```

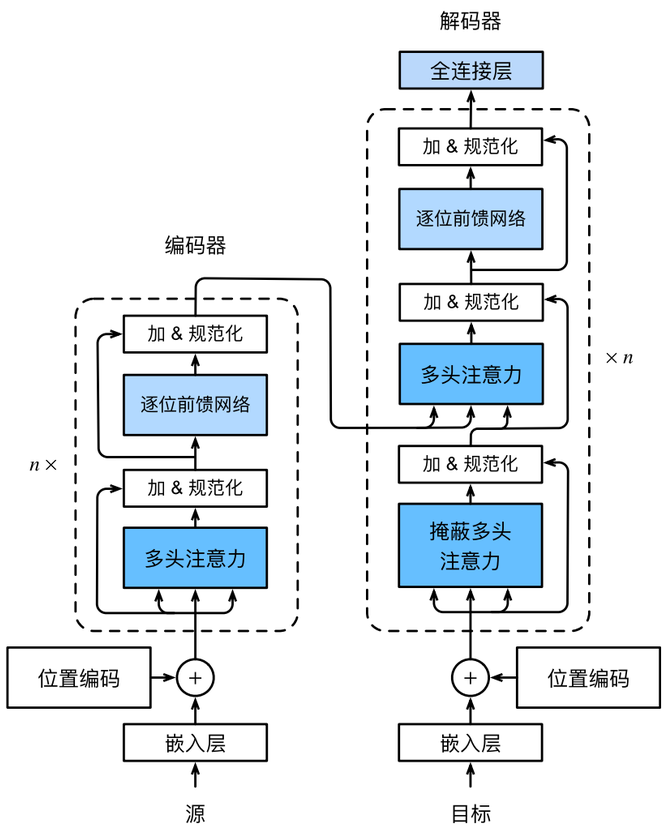
第3节 EncoderDecoder

通用架构

图中的Encoder和Decoder可以随意更换为Seq2Seq网络，如RNN，LSTM，Self-Attention；
图中的state或者说memory是从Encoder胡来的东西，在RNN中，可以是输出序列的平均或加权平均（使用乘积注意力），也可以是最后一个隐藏层输出；在Transformer中，就是Encoder出来的序列

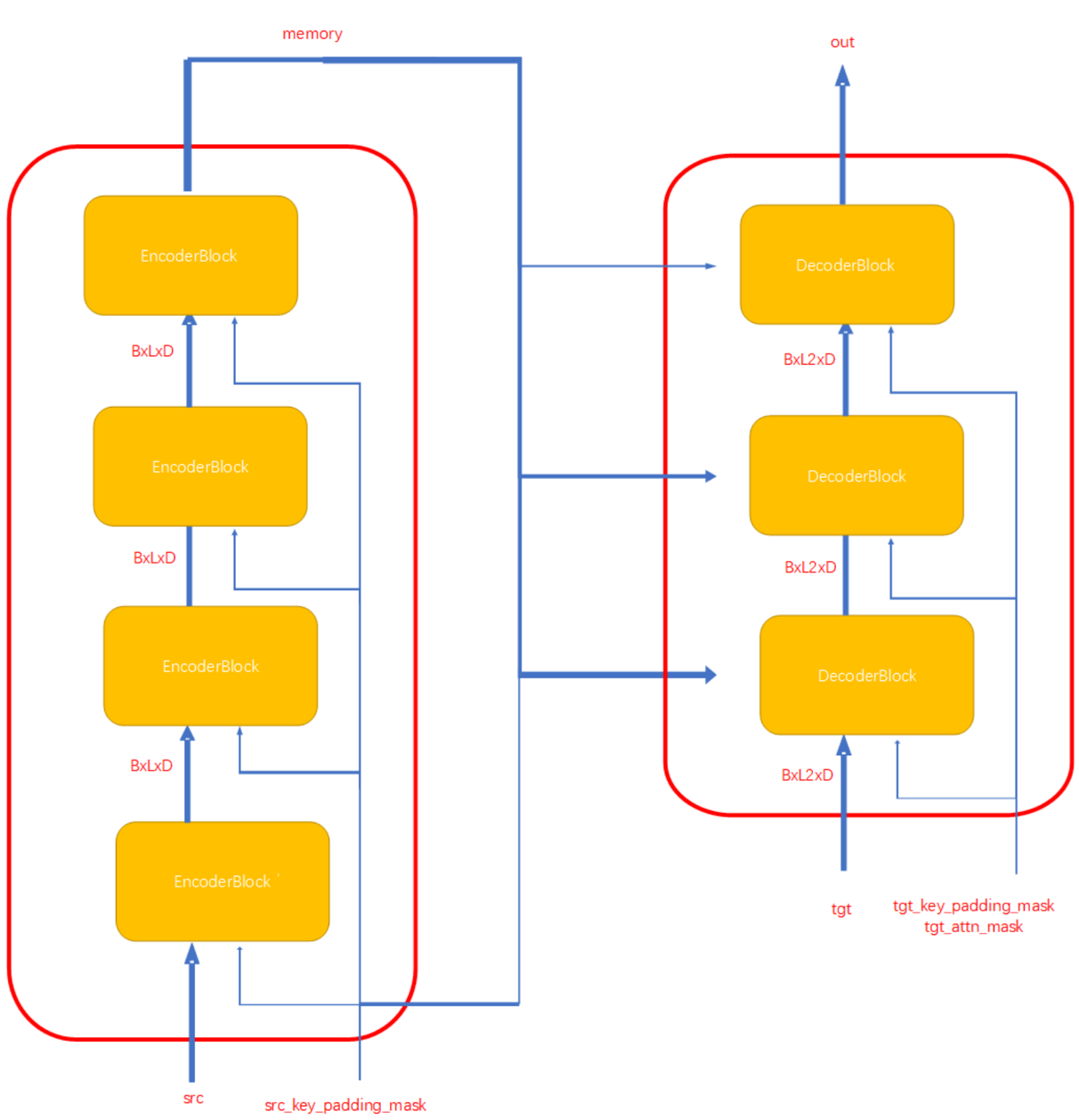


架构细节



组装示意图（4个EncoderBlock，3个DecoderBlock）

Cross-Attention的计算方法：
通常取Encoder的最后一个输出作为Decoder的key和value，
不过也不是一定的，也可以用中间层某个Block的输出作为Decoder中间层某个Block的key和value。



In [17]:
```
class EncoderDecoder(nn.Module):
    def __init__(self,encoderblock,decoderblock,n1,n2):
        super(EncoderDecoder,self).__init__()
        self.encoder=self.block_clone(encoderblock,n1)
        self.decoder=self.block_clone(decoderblock,n2)

    def forward(self,src=None,tgt=None,src_key_padding_mask=None,tgt_key_padding_mask=None,tgt_attn_mask=None):
        X=src
        for layer in self.encoder:
            X=layer(X,src_key_padding_mask)
        memory=X
        X=tgt
        for layer in self.decoder:
            X=layer(tgt=X,memory=memory,src_key_padding_mask=src_key_padding_mask,tgt_key_padding_mask=tgt_key_padding_mask,tgt_attn_mask=tgt_attn_mask)
        return X
    def block_clone(self,block,n):
        return nn.ModuleList([copy.deepcopy(block) for _ in range(n)])
```

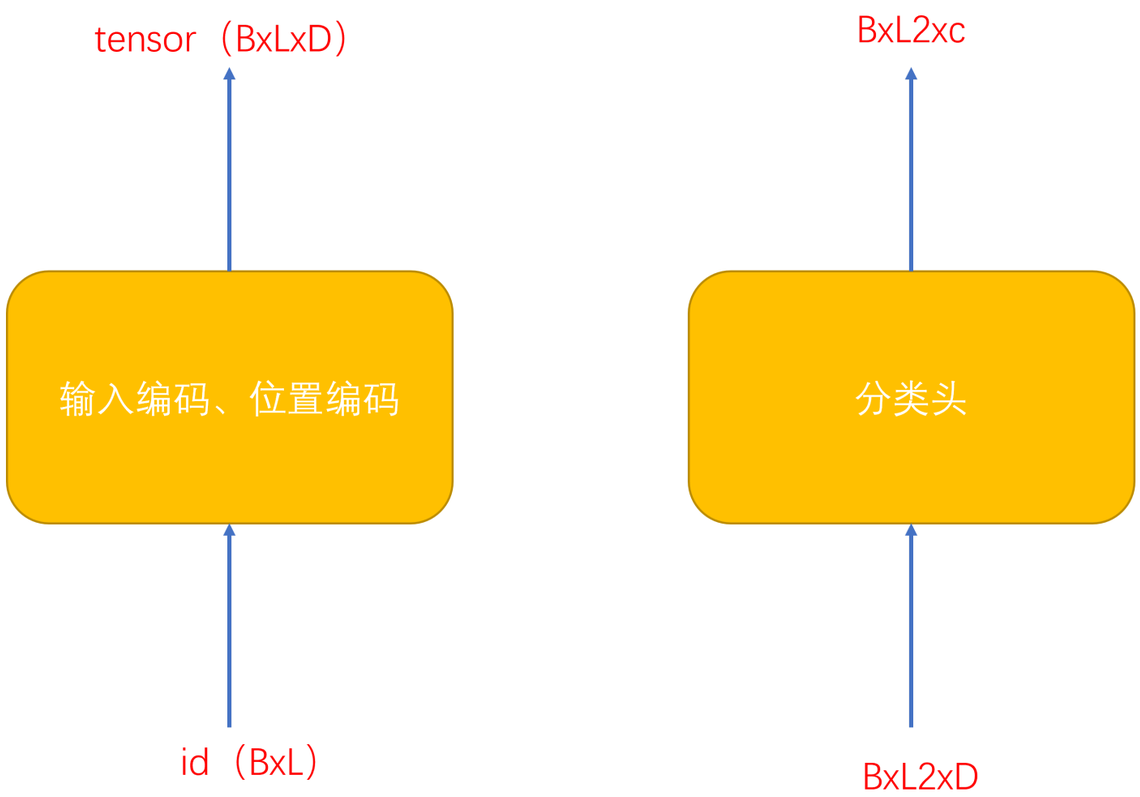
注意，copy.deepcopy是python复制对象的时候，会递归复制这个对象的所有属性，包括引用，
如果里面的方法中使用了某个实例属性，并且这个实例调用过这个方法，会导致这个实例属性

引用了函数栈帧中的变量，这样在复制的时候就会复制失败。
解决方法：
1.重新实例化模型实例，这样实例属性就不会引用到奇怪的地方
2.把实例属性引用到的地方手动变成None

In [18]:
```
# test
encoder=EncoderBlock(d_model=D,num_heads=2,normalized_shape=D,fnn_mid_size=10)
decoder=DecoderBlock(d_model=D,num_heads=2,normalized_shape=D,fnn_mid_size=10)
encoderdecoder=EncoderDecoder(encoderblock=encoder,decoderblock=decoder,n1=3,n2=2)
out6=encoderdecoder(src=input_x,tgt=input_x,src_key_padding_mask=src_key_padding_mask,tgt_key_padding_mask=src_key_padding_mask,tgt_attn_mask=tgt_attn_mask)
out6
```

Out[18]:
```
tensor([[[-4.8789e-02,  3.6814e-01,  1.1982e+00,  7.8317e-01, -8.6184e-01,
          -1.4388e+00],
         [ 3.1358e-01, -1.9900e+00,  1.6731e-01,  1.8317e-01,  6.8637e-01,
           6.3955e-01],
         [-6.2242e-01, -8.2381e-02, -1.5081e+00,  5.4015e-01,  2.7036e-01,
           1.4024e+00],
         [-1.4765e-05, -1.1970e+00,  1.7112e+00,  3.9310e-01, -6.4444e-01,
          -2.6283e-01]],

        [[ 6.2000e-01,  1.2831e-02, -1.0564e+00, -1.1157e+00,  3.8143e-02,
           1.5011e+00],
         [-1.3853e+00,  4.4545e-01,  9.0829e-01,  5.0141e-01, -1.1557e+00,
           6.8585e-01],
         [ 6.0006e-03, -8.1247e-01,  1.3725e+00,  5.7300e-01, -1.4295e+00,
           2.9052e-01],
         [ 9.0596e-01, -1.3936e-01,  5.4616e-01, -1.8614e+00,  6.2522e-01,
          -7.6614e-02]],

        [[-8.0408e-01, -7.6381e-01, -5.8714e-01, -1.2309e-01,  1.7775e+00,
           5.0063e-01],
         [-5.3020e-01, -5.7809e-01, -2.9429e-01,  1.6429e+00, -1.0062e+00,
           7.6581e-01],
         [ 6.1064e-01, -1.2964e+00, -5.9076e-01,  8.9099e-01, -7.3703e-01,
           1.1226e+00],
         [-1.0947e+00, -1.3197e+00,  1.0175e+00,  6.3887e-01, -2.6703e-02,
           7.8474e-01]]], grad_fn=<AddBackward0>)
```

第4节 输入编码、位置编码、分类头



输入编码

注意，这里最后乘以一个系数，是为了和位置编码的数值范围保持一致，
比如位置编码为[8,9,10],单词编码为[0.1，0.2，0.3]，它们相加后相当于
还是位置编码，那么这个模型就只学习位置编码了，跟你的输入半点关系没有

In [19]:
```
class WordEmbedding(nn.Module):
    def __init__(self,num_embeddings,embedding_dim,padding_idx=0):
        super(WordEmbedding,self).__init__()
        self.D=embedding_dim
        self.word_embedding=nn.Embedding(num_embeddings,embedding_dim,padding_idx)  # 没有unk,bos,eos三个特殊字符

    def forward(self,idx):
        return self.word_embedding(idx)*(self.D**0.5)
```

位置编码

下图是一个位置编码矩阵表，t=0所在行表示第0个位置的单词的编码，维度为D，和单词维度一样。
矩阵中每个元素的值由t和i计算而来，具体公式就是下方的公式，当然不会用for遍历每个元素一个一个算，
而是把t和i看成向量，一次性计算，比如t=[[0],[1],[2],[3],[4],[5],[6]],i=[[0,1,2,3,4,5]],然后根据
下方的公式进行计算，注意下方公式分i为奇数偶数两种情况。

	i=0	i=1	i=2	i=3	i=4	i=D
t=0						
t=1						
t=2						
t=3						
t=4						
t=5						
t=T						

$$P_{t,2i} = \sin(\frac{t}{10000^{\frac{2i}{D}}})$$
$$P_{t,2i+1} = \cos(\frac{t}{10000^{\frac{2i}{D}}})$$

In [20]:
```
class PositionEmbedding(nn.Module):
    def __init__(self,max_length,embedding_dim):
        super(PositionEmbedding,self).__init__()
        t=torch.arange(max_length).unsqueeze(-1)
        i=torch.arange(embedding_dim).unsqueeze(0)
        self.W=torch.Tensor(size=[max_length,embedding_dim])  # 不需要学习，不用Parameter类
        # print(t.shape,i.shape,self.W.shape)

        i_even=i[0,0::2]
        self.W[:,0::2]=torch.sin(t/torch.pow(torch.tensor(10000.),2*i_even/embedding_dim))
```

```
        i_odd=i[0,1::2]
        self.W[:,1::2]=torch.cos(t/torch.pow(torch.tensor(10000.),2*i_odd/embedding_dim))

    def __call__(self,idx):
        return self.forward(idx)

    def forward(self,idx):
        return self.W[idx]
```

In [21]:
```
# test
pos=PositionEmbedding(4,3)
pos.W
```

Out[21]:
```
tensor([[0.0000e+00, 1.0000e+00, 0.0000e+00],
        [8.4147e-01, 1.0000e+00, 4.6416e-06],
        [9.0930e-01, 9.9999e-01, 9.2832e-06],
        [1.4112e-01, 9.9998e-01, 1.3925e-05]])
```

In [22]:
```
pos([0,3])
```

Out[22]:
```
tensor([[0.0000e+00, 1.0000e+00, 0.0000e+00],
        [1.4112e-01, 9.9998e-01, 1.3925e-05]])
```

In [23]:
```
pos([[0,3],[0,1]]),pos(torch.tensor([[0,3],[0,1]]))  # idx 一定要用tensor
```

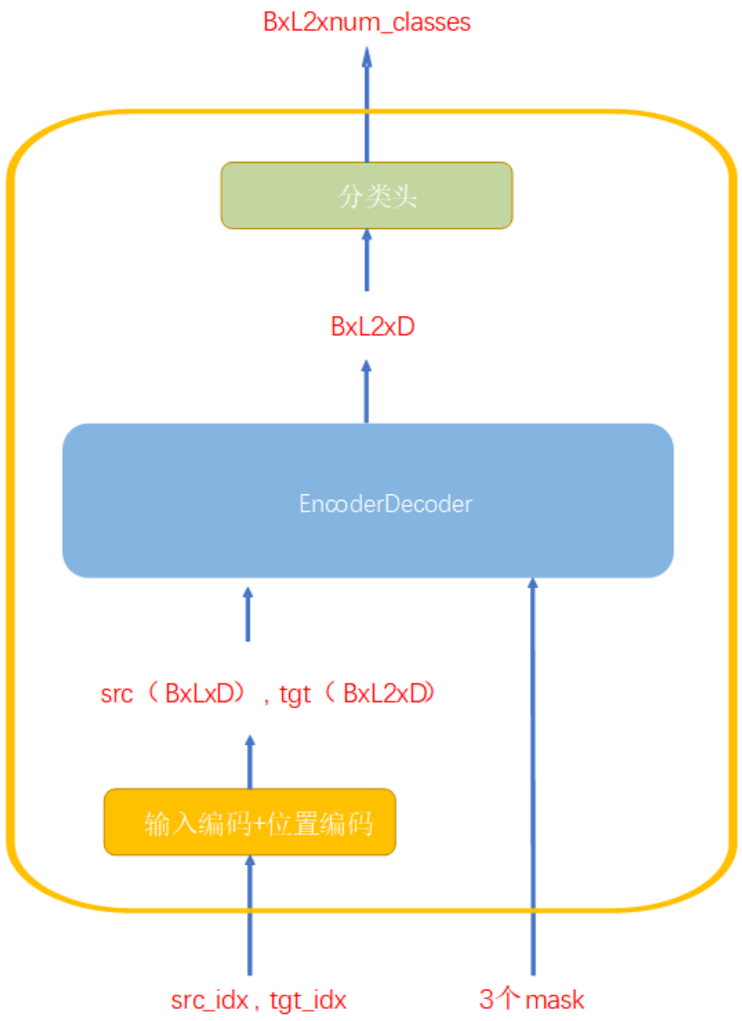
Out[23]:
```
(tensor([0.0000, 1.0000]),
 tensor([[[0.0000e+00, 1.0000e+00, 0.0000e+00],
          [1.4112e-01, 9.9998e-01, 1.3925e-05]],

         [[0.0000e+00, 1.0000e+00, 0.0000e+00],
          [8.4147e-01, 1.0000e+00, 4.6416e-06]]]))
```

分类头

In [24]:
```
class Classifier(nn.Module):
    def __init__(self,hidden_size,num_classes):
        super(Classifier,self).__init__()
        self.linear1=nn.Linear(hidden_size,hidden_size)
        self.activation=nn.Tanh()
        self.linear2=nn.Linear(hidden_size,num_classes)
    def forward(self,x):
        output=self.linear1(x)
        output=self.activation(output)
        output=self.linear2(output)
        return output
```

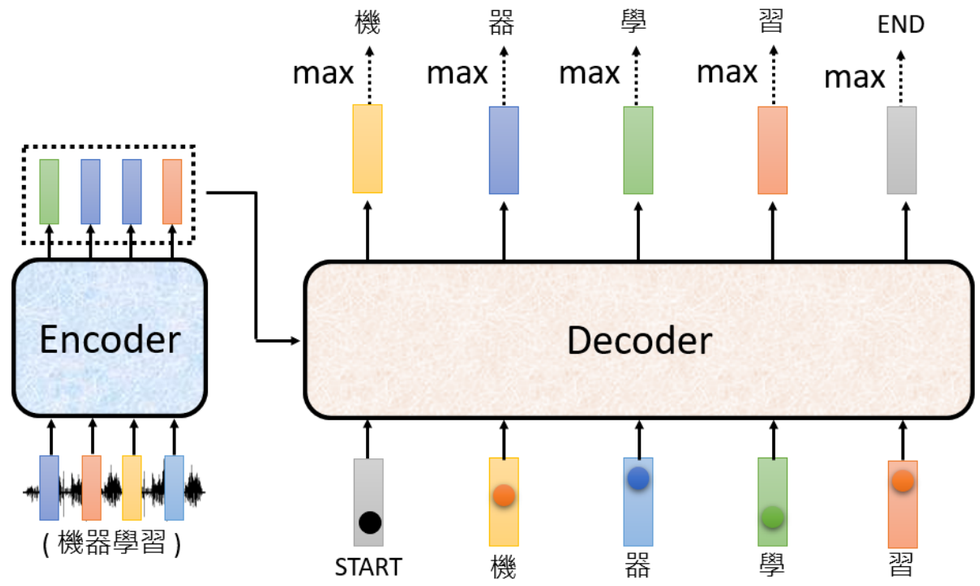
第5节 Transformer总模型

模块	初始化参数	输入	输出
WordEmbedding	num_embeddings , embedding_dim	BxL	BxLxD
PositionEmbedding	max_length , embedding_dim	BxL	BxLxD
EncoderBlock	d_model , d_fnn , num_heads , normalized_shape	src , src_key_padding_mask	BxL1xD
DecoderBlock	d_model , d_fnn , num_heads , normalized_shape	memory , tgt , src_key_padding_mask , tgt_key_padding_mask , tgt_attn_mask	BxL2xD
EcoderDecoder	EncoderBlock , DecoderBlock , n1 , n2	src , tgt , src_key_padding_mask , tgt_key_padding_mask , tgt_attn_mask	BxL2xD
Classifier	hidden_size , num_classes	BxL2xD	BxL2xnum_classes
Transformer	WordEmbedding , PositionEmbedding , EncoderDecoder , Classifier	BxL	BxL2xnum_classes



In [25]:
```
class Transformer(nn.Module):
    """
    使用方法：先定义组件，再初始化Transformer模型
    """
    def __init__(self,src_wordembedding=None,tgt_wordembedding=None,positionembedding=None,encoderdecoder=None,classifier=None):
        super(Transformer,self).__init__()
        self.src_wordembedding=src_wordembedding
        self.tgt_wordembedding=tgt_wordembedding
        self.positionembedding=positionembedding
        self.encoderdecoder=encoderdecoder
        self.classifier=classifier

    def forward(self,src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,tgt_attn_mask):
        src=self.src_wordembedding(src_idx)+self.positionembedding(torch.arange(src_idx.shape[1]))
        tgt=self.tgt_wordembedding(tgt_idx)+self.positionembedding(torch.arange(tgt_idx.shape[1]))
        output=self.encoderdecoder(src,tgt,src_key_padding_mask,tgt_key_padding_mask,tgt_attn_mask)
        output=self.classifier(output)
        return output
```

test



参数设定：

模块	初始化参数
WordEmbedding	num_embeddings=100 , embedding_dim=8
PositionEmbedding	max_length=100 , embedding_dim=8
EncoderBlock	d_model =8, d_fnn=16 , num_heads=2 , normalized_shape=8
DecoderBlock	d_model=8 , d_fnn=16 , num_heads=2 , normalized_shape=8
EcoderDecoder	EncoderBlock , DecoderBlock , n1=3 , n2=3
Classifier	hidden_size=8 , num_classes=100

In [26]:
```
# 基本组件实例化
src_wordembedding=WordEmbedding(100,64)
tgt_wordembedding=WordEmbedding(150,64)
posembedding=PositionEmbedding(50,64)
encoder=EncoderBlock(d_model=64,fnn_mid_size=128,num_heads=2,normalized_shape=64)
decoder=DecoderBlock(d_model=64,fnn_mid_size=128,num_heads=2,normalized_shape=64)
encoderdecoder=EncoderDecoder(encoderblock=encoder,decoderblock=decoder,n1=3,n2=3)
classfier=Classifier(hidden_size=64,num_classes=150)
```

In [27]:
```
# 组装Transformer
model=Transformer(src_wordembedding=src_wordembedding,tgt_wordembedding=tgt_wordembedding,positionembedding=posembedding,encoderdecoder=encoderdecoder,classifier=classfier)
```

In [28]:
```
# 输入构造 BxL=[3,4]
src_idx=torch.tensor([[6,12,44,90],[75,64,26,0],[44,37,0,0]]) # 3句话, valid_len=[4,3,2]
src_valid_len=[4,3,2]
src_key_padding_mask=creat_key_padding_mask(4,src_valid_len)

tgt_idx=torch.tensor([[1,22,24,20],[1,34,36,30],[1,37,0,0]])  # 起始字符bos=1, valid_len=[4,4,2]
tgt_valid_len=[4,4,2]
tgt_key_padding_mask=creat_key_padding_mask(4,tgt_valid_len)
tgt_attn_mask=creat_attn_mask(4)
```

In [29]:
```
# 输出结果的形状
model(src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,tgt_attn_mask).shape
```

Out[29]:
```
torch.Size([3, 4, 150])
```

第6节 带遮掩的损失函数

先简单了解一下官方交叉熵损失的输入形状，后面复现代码时尽量保持一致

`nn.CrossEntropyLoss`对输入序列的形状有一定的要求。下面是关于输入形状的要求：

- 输入形状：`(N, C)`或`(N, C, *)`

其中，

- `N`是批次大小（batch size），
- `C`是类别数（number of classes），
- `*`表示任意额外的维度。

这意味着，输入张量可以是二维的`(N, C)`，也可以是高维的`(N, C, *)`。

具体来说，输入张量的最后两个维度，即形状`(C, *)`，将被视为类别维度和样本维度。

以下是一些示例输入张量的形状：

- `(N, C)`：表示有`N`个样本和`C`个类别，适用于非序列数据的分类问题。
- `(N, C, L)`：表示有`N`个样本，`C`个类别和长度为`L`的序列，适用于序列分类问题。
- `(N, C, H, W)`：表示有`N`个样本，`C`个类别和高度`H`、宽度`W`的图像，适用于图像分类问题。

请注意，对于序列数据，`nn.CrossEntropyLoss`默认会对最后一个维度上的每个时间步骤进行损失计算，并将其平均得到最终损失。如果你想对整个序列的损失进行计算，可以选择使用其他方法，如在时间维度上进行求和或平均。

如果输入张量的形状不符合上述要求，`nn.CrossEntropyLoss`可能会引发形状不匹配的错误。因此，在使用`nn.CrossEntropyLoss`时，请确保输入张量的形状与要求一致。

在PyTorch中，`nn.CrossEntropyLoss`对形状为`(N, C, L)`的序列进行损失计算的实现细节如下：

1. 首先，输入张量`input`和目标张量`target`的形状分别为`(N, C, L)`和`(N, L)`。
2. 内部处理过程会将输入张量`input`转换为形状`(N*L, C)`，以便进行损失计算。这涉及到将序列的维度与类别维度交换，然后使用`view`函数将其调整为二维形状。
3. 目标张量`target`的形状`(N, L)`会被展平为一维形状`(N*L,)`，以与输入张量的形状匹配。
4. 然后，`nn.CrossEntropyLoss`会计算每个时间步骤上的交叉熵损失。它首先将输入张量`input`通过softmax函数，将每个时间步骤上的输出转化为概率分布。这里是对最后一个维度进行softmax。
5. 接下来，对于每个时间步骤，将计算softmax输出与目标张量`target`中对应时间步骤的类别进行比较，计算交叉熵损失。
6. 最后，对所有时间步骤上的损失进行求和或平均，得到最终的序列损失。

使用自己写的CrossEntropyLoss

先实现一个简化版的交叉熵损失，然后基于简化版的交叉熵损失定制子类，
官方是直接把各种功能都封装到CrossEntropyLoss类里面了，注意区别。

实现步骤：
1.实现标准的CrossEntropyLoss
2.对CrossEntropyLoss算出来的loss矩阵掩码
3.在外面对掩码后的loss求平均，注意分母的大小不是矩阵全部的元素个数

In [30]:
```
class CrossEntropyLoss(nn.Module):
    """
    预测：[B,C] 或 [B,C,L]
    标签：[B] 或 [B,L]
    输出：由reduction决定
    """

    def __init__(self,reduction="sum"):
        """
        reduction: 'none' | 'mean' | 'sum'
        """
        super(CrossEntropyLoss, self).__init__()
        self.reduction=reduction

    def forward(self, pred, target):
        B,C=pred.shape[0],pred.shape[1]
        if pred.dim() == 3 and target.dim() == 2:
            pred = pred.transpose(-1, -2).contiguous().view(-1, C)
            target = target.view(-1)

        pred=nn.functional.softmax(pred,dim=-1)  # 官方API把计算概率这步直接封装进来了

        N = pred.shape[0]             # N=B或N=B*L
        if self.reduction=="sum":
            loss = 0
            for i in range(N):
                c = target[i]
                loss -= torch.log(pred[i][c] + 1e-8)
            return loss / N

        if self.reduction=="none":
            loss = []
            for i in range(N):
                c = target[i]
                loss.append(-torch.log(pred[i][c] + 1e-8))
            # 将多个tensor合并，注意这个操作必须实现了反向传播算法，不要用torch.tensor(loss)创建一个没反向算子的新张量
            loss=torch.stack(loss)
            loss=loss.view(B,-1)     # BxL
            return loss

        if self.reduction=="mean":
            loss = []
            for i in range(N):
                c = target[i]
                loss.append(-torch.log(pred[i][c] + 1e-8))
            # 将多个tensor合并，注意这个操作必须实现了反向传播算法，不要用torch.tensor(loss)创建一个没反向算子的新张量
            loss=torch.stack(loss)
            loss=loss.view(B,-1)     # BxL
            return loss.mean()
```

In [31]:
```
class MaskedCrossEntropyLoss_V1(CrossEntropyLoss):
    def __init__(self,mask):
        super(MaskedCrossEntropyLoss_V1,self).__init__()
        self.mask=mask
    def forward(self,pred,label):
        self.reduction="none"
        loss=super().forward(pred.transpose(-1,-2),label)  # 输入序列要求形状为 BxcxL ， c是类别数
        loss=torch.masked_fill(loss,self.mask,0)
        return loss
```

手写API测试

In [32]:
```
# test data
pred=torch.rand(4,3,8)      # 4个句子，每个句子包含3个单词，每个单词预测分布是8维向量，就是预测8个类别
label=torch.ones(4,3).long()
mask=torch.tensor([[True,True,True],[True,True,False],[True,False,False],[False,False,False]])
label=torch.masked_fill(label,~mask,0)  # 简单起见，这里除了pad的标签是0，其他标签都设为1
print(label)  # 打印出4句话，每句话3个单词的预测标签，除了pad词预测的标签为0，其他词预测的词id都是1
```

```
tensor([[1, 1, 1],
        [1, 1, 0],
        [1, 0, 0],
        [0, 0, 0]])
```

In [33]:
```
# test
loss_fn1=MaskedCrossEntropyLoss_V1(~mask)
loss1=loss_fn1(pred,label)
loss1,loss1.sum()/6
```

Out[33]:
```
(tensor([[2.4428, 2.6800, 2.1702],
         [1.7623, 2.0221, 0.0000],
         [1.9997, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000]]),
 tensor(2.1795))
```

基于官方API改造

In [34]:
```
class MaskedCrossEntropyLoss(nn.CrossEntropyLoss):
    def __init__(self,mask):
        super(MaskedCrossEntropyLoss,self).__init__()
        self.mask=mask
    def forward(self,pred,label):
        self.reduction="none"
        loss=super().forward(pred.transpose(-1,-2),label)  # 输入序列要求形状为 BxcxL ， c是类别数
        loss=torch.masked_fill(loss,self.mask,0)
        return loss
```

In [35]:
```
# test
loss_fn2=MaskedCrossEntropyLoss(~mask)
loss2=loss_fn2(pred,label)
loss2,loss2.sum()/6
```

Out[35]:
```
(tensor([[2.4428, 2.6800, 2.1702],
         [1.7623, 2.0221, 0.0000],
         [1.9997, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000]]),
 tensor(2.1795))
```

直接利用官方写好的ignore_index参数直接忽略标签为0的损失

In [36]:
```
loss_fn3=nn.CrossEntropyLoss(ignore_index=0,reduction="none")
loss3=loss_fn3(pred.transpose(-1,-2),label)
loss3,loss3.sum()/6
```

Out[36]:
```
(tensor([[2.4428, 2.6800, 2.1702],
         [1.7623, 2.0221, 0.0000],
         [1.9997, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000]]),
 tensor(2.1795))
```

直接利用官方写好的weight参数，令pad字符的预测损失权重为0

In [37]:
```
weight = torch.tensor([0.,1,1,1,1,1,1,1])
loss_fn4=nn.CrossEntropyLoss(weight=weight,reduction="none")
loss4=loss_fn4(pred.transpose(-1,-2),label)
loss4,loss4.sum()/6
```

Out[37]:
```
(tensor([[2.4428, 2.6800, 2.1702],
         [1.7623, 2.0221, 0.0000],
         [1.9997, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000]]),
 tensor(2.1795))
```

官方API的reduction参数可以正确求得掩码后的均值

注意，前面自己手写的，只能在外面求和后，除以正确的分母
后面这个东西就用官方的了，自己实现的整起来有点麻烦

In [38]:
```
loss_fn5=nn.CrossEntropyLoss(ignore_index=0,reduction="mean")
loss5=loss_fn5(pred.transpose(-1,-2),label)
loss5
```

Out[38]:
```
tensor(2.1795)
```

第7节 翻译质量评估-BLEU

首先我们从模型也就是transformer得到output张量，也就是BxLxc的输出，c是类别数，
另外我们从dataloader里拿到两个东西，分别是真实标签label以及tgt_key_padding_mask，

形状均为BxL，由于label和tgt每条句子长度是一致的，它们的填充和掩码其实都是一回事，
有了这3样东西我们才能计算出Blue-score(算的是corpus_blue)

第一步：Output转换

将output转换为类似[["我","叫","张","飞","EOS"] , ["是","的","EOS"] , ["怎","么","了","EOS"]] 的candi_sents;
将label转换为类似[[["我","叫","张","三", "EOS"] , ["是","滴", "EOS"] , ["怎","么","啦", "EOS"]]] 的refer_sents;
注意，下面这个示例，为了简便起见，直接把output当成概率分布矩阵来看，实际上我们上面实现的transformer输出的output还没有进行softmax归一化，还不是概率分布矩阵。但是到了后面真正要用的时候，我们再里面会加上softmax处理

In [39]:
```
def trans_func(output, label, tgt_key_padding_mask, vocab):
    """
    :param output: BxLxc的概率预测张量
                   [[[0.1, 0.2, 0.7], [0.3, 0.5, 0.2], [0.2, 0.4, 0.4]],
                   [[0.4, 0.1, 0.5], [0.2, 0.3, 0.5], [0.6, 0.2, 0.2]]]
    :param label: BxL的标签
                   [[[0], [1], [2]], [[1], [0], [2]]]
    :param seq_mask: [[1, 1, 1], [1, 1, 0]]
    :param vocab:  {0: 'apple', 1: 'banana', 2: 'orange'}

    :return:
           Candidate: [['orange', 'banana', 'banana'], ['orange', 'orange']]
           Reference: [[['apple', 'banana', 'orange']], [['banana', 'apple']]]
    """
    seq_mask = tgt_key_padding_mask
    pred = torch.argmax(output, dim=-1)
    cand, ref_list = [], []
    for i in range(pred.shape[0]):
        token_list = []
        for j in range(pred.shape[1]):
            if seq_mask[i][j] == 0:
                break
            token_list.append(vocab[pred[i][j].item()])
        cand.append(token_list)

    for i in range(label.shape[0]):
        token_list = []
        for j in range(label.shape[1]):
            if seq_mask[i][j] == 0:
                break
            token_list.append(vocab[label[i][j].item()])

        ref_list.append([token_list])

    return cand, ref_list
```

In [40]:
```
# test
output = torch.tensor([[[0.1, 0.2, 0.7], [0.3, 0.5, 0.2], [0.2, 0.4, 0.4]],
                       [[0.4, 0.1, 0.5], [0.2, 0.3, 0.5], [0.6, 0.2, 0.2]]])

tgt = torch.tensor([[[0], [1], [2]], [[1], [0], [2]]])

tgt_key_padding_mask =  torch.tensor([[1, 1, 1], [1, 1, 0]])

vocab = {0: 'apple', 1: 'banana', 2: 'orange'}
```

In [41]:
```
candicate, reference = trans_func(output, tgt, tgt_key_padding_mask, vocab)
print("Candidate:", candicate)
print("Reference:", reference)
```

```
Candidate: [['orange', 'banana', 'banana'], ['orange', 'orange']]
Reference: [[['apple', 'banana', 'orange']], [['banana', 'apple']]]
```

transformer输出的output不是概率分布矩阵，所以后面我们有个trans_func，会在里面加个softmax
另外需要传进去一个vocab进行转换比较麻烦，后面写的不进行转换了，但是对于candicate=[tensor(1),tensor(2)]这样的进行切ngram后再求交集，无法匹配成功，里面的元素不能是tensor，前面的字符类型可以，后面测试了以下，数值类型也行，tensor反正是不行

第二步：计算corpus bleu

首先对于每个句子，都要计算下面的参数

准确率的分子分母	1-gram	2-gram	3-gram	4-gram
match_ngram 翻对的ngram				
candi_ngram 翻译的ngram数目				

惩罚项公式的两个长度	candi=["怎","么","了"]，refer=[["咋","了"]]
bp_c , len(candicate)	3
bp_r , len(closest_reference)	2

对于多批句子，计算corpus bleu的步骤如下

```
self.match_ngram = {}
self.candi_ngram = {}
self.bp_c=0
self.bp_r=0
for B in Batch(遍历每个Batch):
    for candi,refer in B(遍历当前Batch中的每个句子):
        对于当前句子，计算上面表格中所有需要计算的量
        将上面的计算结果累加到:
            self.match_ngram，self.candi_ngram，self.bp_c，self.bp_r
self.accumulate():进行bleu分数计算
```

上面计算过程中需要实现的几个关键函数

1.get_ngram(sent, k_size): k_size就是kgram的k，这个函数把一个句子按k切成kgram
2.modified_precision(cand_ngram, refs_ngram, k_size):计算pk的分子和分母
3.bp_count(self,cand_sent, refs_sent): 计算惩罚项中的候选句和参考句长度
注意，2和3都是针对单句话操作的，操作结果放到累加器中
单个句子的输入： cand是list，refer是list of list
一个batch_size的输入：对上面再套一层list

In [42]:
```
class Metric:
    """
    所有评价的父类，必须实现下面3个方法
    """
    @abc.abstractmethod
    def update(self,*args,**kwargs):
        raise NotImplementedError

    @abc.abstractmethod
    def accumulate(self):
        raise NotImplementedError

    @abc.abstractmethod
    def reset(self):
        raise NotImplementedError

    def score(self):
        return self.accumulate()

class Bleu(Metric):

    def __init__(self,n_size,weights=None):
        super(Bleu, self).__init__()
        if not weights:
            weights = [1 / n_size for _ in range(n_size)]
        self.weights = weights
        self.n_size = n_size

        self.match_ngram = {}
        self.candi_ngram = {}
        self.bp_r = 0
        self.bp_c = 0

        for k in range(n_size):
            self.match_ngram[k]=0
            self.candi_ngram[k]=0

    # 切词，切出ngram
    def get_ngram(self,sent, k_size):
```

```
        """
        :param sent: ["我","叫","张","三"]
        :param n_size: 2
        :return: [["我","叫"],["叫","张"],["张","三"]]
        """
        ngram_list = []
        for left in range(len(sent) - k_size):
            ngram_list.append(sent[left : left + k_size + 1])
        ngram_list = [tuple(gram) for gram in ngram_list]
        return ngram_list

    # 针对某个k_size的ngram计算准确率的分子分母，并送入累加器
    def kgram_precision(self,cand_kgram, refs_kgram, k_size):
        """
        :param cand_kgram: [("我","叫"),("叫","张"),("张","三")]
        :param refs_kgram: [[("我","叫"),("叫","张"),("张","三")],[("我","是"),("是","张"),("张","三")]]
        """
        # 对切好词的列表进行统计
        cand_counts = Counter(cand_kgram)

        # 求出参考答案的kgram特殊并集
        refers=[Counter(ref) for ref in refs_kgram]
        refer_counts = Counter()
        for key in cand_counts.keys():
            max_value=0
            for ref in refers:
                max_value=max(max_value,ref[key])
            refer_counts[key]=max_value

        # 求交集
        matched_counts = {ngram: min(count, refer_counts[ngram]) for ngram, count in cand_counts.items()}

        # 计算分子
        match_size = sum(matched_counts.values())
        # 计算分母，注意分母不能为0
        cand_size = max(1, sum(cand_counts.values()))

        # 把分子和分母送入累加器中
        self.match_ngram[k_size] += match_size
        self.candi_ngram[k_size] += cand_size

    # 计算惩罚项中的长度，并送入累加器
    def bp_count(self,cand, refs):
        cand_len=len(cand)
        ref_lens = (len(refer) for refer in refs)
        closest_ref_len = min(ref_lens, key=lambda ref_len: (abs(ref_len - cand_len), ref_len))
        self.bp_c+=cand_len
        self.bp_r+=closest_ref_len

    # 上面的组件都已经准备完毕，开始计算一个句子的各项表格中的数值，并送入累加器中
    def _update(self,cand,refers):
        # 对于每个k，计算pk的分子分母
        for k in range(self.n_size):
            # 切词
            cand_kgram=self.get_ngram(cand,k)
            refer_kgram=[]
            for refer in refers:
                refer_kgram.append(self.get_ngram(refer,k))
            # 计算pk的分子分母，并送入累加器
            self.kgram_precision(cand_kgram,refer_kgram,k)
        # 根据整个句子，计算bp_c和bp_r
        self.bp_count(cand,refers)

    # 计算一个batch_size中的句子的各项表格中的数值，并送入累加器中
    def update(self,candicate,reference):
        batch_size=len(candicate)
        for i in range(batch_size):
            self._update(candicate[i],reference[i])

    # 根据累加器中的值，计算bleu分数
    def accumulate(self):
        # 计算n次的准确率
        p=[]
        for i in range(self.n_size):
            pi=self.match_ngram[i]/self.candi_ngram[i]
            if pi == 0:
                p.append(sys.float_info.min)
            else:
                p.append(pi)
        # 计算惩罚项
        bp = math.exp(1 - self.bp_c / self.bp_r) if self.bp_c < self.bp_r else 1
        # 计算bleu分数
        score = (w_i * math.log(p_i) for (w_i,p_i) in zip(self.weights,p))
        score = bp * math.exp(math.fsum(score))
        return score

    def reset(self):
        self.match_ngram = {}
        self.candi_ngram = {}
        self.bp_r = 0
        self.bp_c = 0
        for k in range(self.n_size):
            self.match_ngram[k]=0
            self.candi_ngram[k]=0
```

测试一：计算短句子，比较上面实现的和nltk的计算结果是否相同

In [43]:
```
Candidate = [['orange', 'banana', 'banana'], ['orange', 'orange']]
Reference = [[['apple', 'banana', 'orange']], [['banana', 'apple']]]
```

In [44]:
```
corpus_bleu_=Bleu(4)
corpus_bleu_.update(candicate,reference)
corpus_bleu_.score()
```

Out[44]:
```
1.4488496539373276e-231
```

In [45]:
```
nltk.translate.bleu_score.corpus_bleu(reference,candicate)
```

Out[45]:
```
1.4488496539373276e-231
```

测试二：计算长句子，比较上面实现的和nltk的计算结果是否相同

In [46]:
```
hyp1 = ['It', 'is', 'a', 'guide', 'to', 'action', 'which','ensures', 'that', 'the', 'military', 'always','obeys', 'the', 'commands', 'of', 'the', 'party']
ref1a = ['It', 'is', 'a', 'guide', 'to', 'action', 'that','ensures', 'that', 'the', 'military', 'will', 'forever','heed', 'Party', 'commands']
ref1b = ['It', 'is', 'the', 'guiding', 'principle', 'which','guarantees', 'the', 'military', 'forces', 'always','being', 'under', 'the', 'command', 'of', 'the', 'Party']
ref1c = ['It', 'is', 'the', 'practical', 'guide', 'for', 'the','army', 'always', 'to', 'heed', 'the', 'directions','of', 'the', 'party']
hyp2 = ['he', 'read', 'the', 'book', 'because', 'he', 'was','interested', 'in', 'world', 'history']
ref2a = ['he', 'was', 'interested', 'in', 'world', 'history','because', 'he', 'read', 'the', 'book']
list_of_references = [[ref1a, ref1b, ref1c], [ref2a]]
hypotheses = [hyp1, hyp2]
```

In [47]:
```
corpus_bleu_=Bleu(4)
corpus_bleu_.update(hypotheses,list_of_references)
corpus_bleu_.score()
```

Out[47]:
```
0.5920778868801042
```

In [48]:
```
nltk.translate.bleu_score.corpus_bleu(list_of_references,hypotheses)
```

Out[48]:
```
0.5920778868801042
```

如何使用corpus bleu？

关于corpus的3个level：sentence level ， batch level，dataset level
方案一：计算 batch level 的 corpus bleu， 然后求平均
方案二：计算 dataset level 的 corpus bleu

一般我们采用方案二

W 机器翻译计算corpus bleu分数时，corpus bleu是对一个batch size的数据计算，还是对整个dataset的数据计算

机器翻译中计算corpus BLEU分数时，通常是针对整个数据集（dataset）进行计算，而不是仅针对一个批次（batch size）的数据计算。

Corpus BLEU分数是用于评估整个语料库的机器翻译结果与参考翻译之间的相似度和质量。它考虑了整个数据集中的所有句子，并将所有的机器翻译结果与对应的参考翻译进行比较。通过对整个数据集进行计算，可以得到一个综合的评估结果，用于评估机器翻译系统在整个数据集上的性能。

因此，在计算corpus BLEU分数时，需要将整个数据集的机器翻译结果和对应的参考翻译作为输入，并计算整个数据集的BLEU分数。这样可以更全面地评估机器翻译系统在实际应用中的表现。

需要注意的是，如果数据集过大，计算整个数据集的BLEU分数可能会比较耗时。在实际应用中，可以根据具体需求和计算资源的限制，选择对部分数据进行计算或使用采样方法来近似计算整个数据集的BLEU分数。

对上面的Bleu加入光滑

之所以上面实现没有加任何东西，是为了对比计算结果，后面我们使用的还是要加入光滑的
修改的地方：match_size = sum(matched_counts.values())+1

In [49]:
```
class Metric:
    """
    所有评价的父类，必须实现下面3个方法
    """
    @abc.abstractmethod
    def update(self,*args,**kwargs):
        raise NotImplementedError

    @abc.abstractmethod
    def accumulate(self):
        raise NotImplementedError

    @abc.abstractmethod
    def reset(self):
        raise NotImplementedError

    def score(self):
        return self.accumulate()

class Bleu(Metric):

    def __init__(self,n_size,weights=None):
        super(Bleu, self).__init__()
        if not weights:
            weights = [1 / n_size for _ in range(n_size)]
        self.weights = weights
        self.n_size = n_size

        self.match_ngram = {}
        self.candi_ngram = {}
        self.bp_r = 0
        self.bp_c = 0

        for k in range(n_size):
            self.match_ngram[k]=0
            self.candi_ngram[k]=0

    # 切词，切出ngram
    def get_ngram(self,sent, k_size):
        """
        :param sent: ["我","叫","张","三"]
        :param n_size: 2
        :return: [["我","叫"],["叫","张"],["张","三"]]
        """
        ngram_list = []
        for left in range(len(sent) - k_size):
            ngram_list.append(sent[left : left + k_size + 1])
        ngram_list = [tuple(gram) for gram in ngram_list]
        return ngram_list

    # 针对某个k_size的ngram计算准确率的分子分母，并送入累加器
    def kgram_precision(self,cand_kgram, refs_kgram, k_size):
        """
        :param cand_kgram: [("我","叫"),("叫","张"),("张","三")]
        :param refs_kgram: [[("我","叫"),("叫","张"),("张","三")],[("我","是"),("是","张"),("张","三")]]
        """
        # 对切好词的列表进行统计
        cand_counts = Counter(cand_kgram)

        # 求出参考答案的kgram特殊并集
        refers=[Counter(ref) for ref in refs_kgram]
        refer_counts = Counter()
        for key in cand_counts.keys():
            max_value=0
            for ref in refers:
                max_value=max(max_value,ref[key])
            refer_counts[key]=max_value

        # 求交集
        matched_counts = {ngram: min(count, refer_counts[ngram]) for ngram, count in cand_counts.items()}

        # 计算分子
        match_size = sum(matched_counts.values())+1
        # 计算分母，注意分母不能为0
        cand_size = max(1, sum(cand_counts.values()))

        # 把分子和分母送入累加器中
        self.match_ngram[k_size] += match_size
        self.candi_ngram[k_size] += cand_size

    # 计算惩罚项中的长度，并送入累加器
    def bp_count(self,cand, refs):
        cand_len=len(cand)
        ref_lens = (len(refer) for refer in refs)
        closest_ref_len = min(ref_lens, key=lambda ref_len: (abs(ref_len - cand_len), ref_len))
        self.bp_c+=cand_len
        self.bp_r+=closest_ref_len

    # 上面的组件都已经准备完毕，开始计算一个句子的各项表格中的数值，并送入累加器中
    def _update(self,cand,refers):
        # 对于每个k，计算pk的分子分母
        for k in range(self.n_size):
            # 切词
            cand_kgram=self.get_ngram(cand,k)
            refer_kgram=[]
            for refer in refers:
                refer_kgram.append(self.get_ngram(refer,k))
            # 计算pk的分子分母，并送入累加器
            self.kgram_precision(cand_kgram,refer_kgram,k)
        # 根据整个句子，计算bp_c和bp_r
        self.bp_count(cand,refers)

    # 计算一个batch_size中的句子的各项表格中的数值，并送入累加器中
    def update(self,candicate,reference):
        batch_size=len(candicate)
        for i in range(batch_size):
            self._update(candicate[i],reference[i])

    # 根据累加器中的值，计算bleu分数
    def accumulate(self):
        # 计算n次的准确率
        p=[]
        for i in range(self.n_size):
```

```
            pi=self.match_ngram[i]/self.candi_ngram[i]
            if pi == 0:
                p.append(sys.float_info.min)
            else:
                p.append(pi)
        # 计算惩罚项
        bp = math.exp(1 - self.bp_c / self.bp_r) if self.bp_c < self.bp_r else 1
        # 计算bleu分数
        score = (w_i * math.log(p_i) for (w_i,p_i) in zip(self.weights,p))
        score = bp * math.exp(math.fsum(score))
        return score

    def reset(self):
        self.match_ngram = {}
        self.candi_ngram = {}
        self.bp_r = 0
        self.bp_c = 0
        for k in range(self.n_size):
            self.match_ngram[k]=0
            self.candi_ngram[k]=0
```

第8节 训练框架设计

In [50]:

```
class Transformer_Runner():
    def __init__(self, model=None, loss_fn=None, optimizer=None, metric=None):

        self.model = model
        self.loss_fn = loss_fn
        self.optimizer = optimizer
        self.metric = metric

        self.train_step_loss = []  # 每步的训练损失
        self.dev_stride_loss = []  # 每stride步的测试损失
        self.dev_stride_score = []  # 每stride步的测试分数
        self.best_score = 0  # 记录评估时最好的得分

    def train(self, train_loader, dev_loader=None, num_epochs=1, log_stride=1):
        step = 0
        total_steps = num_epochs * len(train_loader)

        for epoch in range(num_epochs):
            for inputs,label in train_loader:
                src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask=inputs
                output = self.model(src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask)
                loss = self.loss_fn(output.transpose(-1,-2), label)

                # 记录训练损失，并打印
                self.train_step_loss.append((step,loss.item()))
                if step % log_stride == 0 or step == total_steps - 1:
                    print("[Train]  epoch:{}/{} step:{}/{} loss:{:.4f}".format(epoch, num_epochs, step, total_steps,
                                                                               loss.item()))
                # 反向传播，更新参数
                loss.backward()
                self.optimizer.step()
                self.optimizer.zero_grad()

                # 记录并打印评估的损失和分数
                if dev_loader:
                    if step % log_stride == 0 or step == total_steps - 1:
                        dev_loss,dev_score = self.evaluate(dev_loader)
                        self.dev_stride_loss.append((step,dev_loss))
                        self.dev_stride_score.append((step,dev_score))
                        print("[Evaluate]  loss:{:.4f} score:{:.5f} ".format(dev_loss,dev_score))

                        # 记录最优分数，并保存模型
                        if dev_score > self.best_score:
                            self.best_score = dev_score
                            saves_path = os.getcwd()
                            saves_name = f"step{step}-score{self.best_score:.5f}.params"
                            self.save(path=os.path.join(saves_path, saves_name))
                            print("Best model has been updated and saved!")

                # 步数+1
                step += 1
        # 画图
        print("done!")
        self.plot()

    @torch.no_grad()
    def evaluate(self, dev_loader):

        self.model.eval()
        self.metric.reset()

        total_loss = 0

        for inputs,label in dev_loader:
            src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask=inputs
            output = self.model(src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask)
            loss = self.loss_fn(output.transpose(-1,-2), label)        # 先求batch level的loss平均
            total_loss += loss.item()

            candicate,reference=self.trans_func(output,label,tgt_key_padding_mask)
            # print("候选翻译",candicate,"\n")
            # print("参考翻译",reference,"\n")
            self.metric.update(candicate,reference)

        dev_loss = total_loss / len(dev_loader)      # 再对所有batch的loss平均求平均
        dev_score = self.metric.score()
        return dev_loss,dev_score

    def trans_func(self,output,label,tgt_key_padding_mask):
        """
        实际上并不需要转换成字符，这里由于传入字典比较多余且麻烦，就不进行转换了
        """
        seq_mask = ~tgt_key_padding_mask
        pred = torch.argmax(F.softmax(output,dim=-1), dim=-1)
        cand, ref_list = [], []

        for i in range(pred.shape[0]):
            token_list = []
            for j in range(pred.shape[1]):
                if seq_mask[i][j] == 0:
                    break
                token_list.append(pred[i][j].item())
            cand.append(token_list)

        for i in range(label.shape[0]):
            token_list = []
            for j in range(label.shape[1]):
                if seq_mask[i][j] == 0:
                    break
                token_list.append(label[i][j].item())

            ref_list.append([token_list])

        return cand, ref_list


    def plot(self):
        plt.figure(figsize=(8, 4))
        plt.clf()

        # 损失函数曲线
        plt.subplot(1,2,1)
        plt.xlabel("step")
        plt.ylabel("loss")

        stride=1              # 根据自己的需求调整
        train_stride_loss=self.train_step_loss[::stride]
        train_x=[x[0] for x in train_stride_loss]
        train_y=[x[1] for x in train_stride_loss]
        plt.plot(train_x,train_y,label="Train loss")

        test_x=[x[0] for x in self.dev_stride_loss]
        test_y=[x[1] for x in self.dev_stride_loss]
        plt.plot(test_x,test_y,label="Dev loss")
        plt.legend()

        # 评估曲线
        if len(self.dev_stride_loss) !=0:
            plt.subplot(1,2,2)
            plt.xlabel("step")
            plt.ylabel("score")
            plt.plot(*zip(*self.dev_stride_score),label="BLEU Score")
```

```
            plt.legend()

        # 显示图像
        plt.tight_layout()
        plt.draw()


    def save(self, path):
        torch.save(self.model.state_dict(), path)

    def load(self, path):
        state_dict = torch.load(path)
        self.model.load_state_dict(state_dict)
```

第9节 数据集处理

数据集来源：http://www.manythings.org/anki/

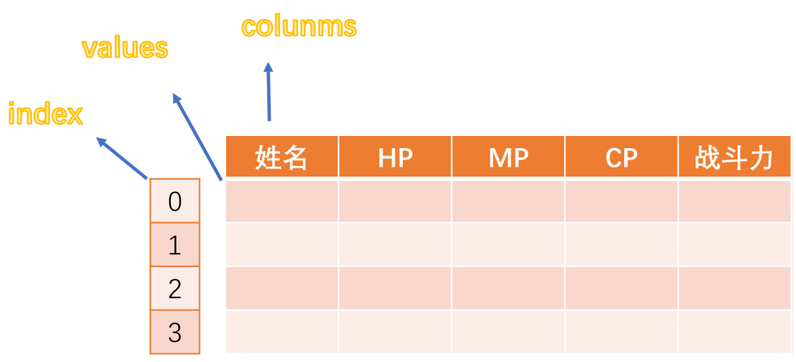
- Kannada - English kan-eng.zip (157)
- Kapampangan - English pam-eng.zip (1157)
- Khasi - English kha-eng.zip (1567)
- Khmer - English khm-eng.zip (798)
- Korean - English kor-eng.zip (5794)
- Latvian - English lvs-eng.zip (1843)
- Lithuanian - English lit-eng.zip (2140)
- Low German (Low Saxon) - English nds-eng.zip (3149)
- Macedonian - English mkd-eng.zip (62633)
- Malay - English zsm-eng.zip (401)
- Malayalam - English mal-eng.zip (613)
- Mandarin Chinese - English cmn-eng.zip (29476)
- Maori - English mri-eng.zip (208)
- Marathi - English mar-eng.zip (46966)
- North Moluccan Malay - English max-eng.zip (175)
- Northern Kurdish (Kurmancî) - English kmr-eng.zip (437)
- Norwegian Bokmål - English nob-eng.zip (7391)
- Nuer - English nus-eng.zip (1522)
- Odia (Oriya) - English ori-eng.zip (102)
- Persian - English pes-eng.zip (3103)
- Polish - English pol-eng.zip (47159)
- Portuguese - English por-eng.zip (188739)
- Romani - English rom-eng.zip (572)
- Romanian - English ron-eng.zip (14237)
- Russian - English rus-eng.zip (467119)

- These are sorted by length, with the s

Would you like to help?

- There are over 800,000 good, proofre audio files. *(763,276 / 893,849 on Februa*
- If you are a native speaker of one of Project and help by adding translatio translated.
- If your native language isn't here, the
- See a random selection of proofread

pandas结构



创建：pd.DataFrame(index=list-array,columns=list-array,values=list-array)
pd.DataFrame({"姓名":list-array,"HP":list-array})

读取：pd.read_csv(路径,sep=分隔符,encoding=编码,index_col=列id,header=行id,names=手动创建标签列表当columns)

df[ ] : ①.取行操作：标签或数字切片 ②.取列操作：单个标签或标签列表

df.loc[标签行：标签列] ，这里的标签可以是离散的列表，也可以是连续的切片
df.iloc[数字行：数字列]，这里的数字可以是离散的列表，也可以是连续的切片

布尔索引：一维索引：list-array-series 二维索引：array-dataframe 均返回一张新表

布尔索引	一维	二维
[]	仅用于筛选行	将不满足的元素变成NAN
loc[]	可以筛选行或列	\
iloc[]	可以筛选行或列	\

第1步：读取数据

In [51]:
```
data=pd.read_csv(r"C:\Users\Administrator\Desktop\Open1507Lab\data\dataset\cmn.txt",sep="\t",usecols=[0, 1],names=["English","Chinese"])
data.head()
```

Out[51]:

	English	Chinese
0	Hi.	嗨。
1	Hi.	你好。
2	Run.	你用跑的。
3	Stop!	住手!
4	Wait!	等等!

In [52]:
```
data[-5:]
```

Out[52]:

	English	Chinese
29471	If you don't want to put on sunscreen, that's ...	你不想涂防晒霜是你的问题，但是晒伤了不要来抱怨。
29472	Even now, I occasionally think I'd like to see...	即使是现在，我偶尔还是想见到你。不是今天的你，而是我记忆中曾经的你。
29473	It's very easy to sound natural in your own na...	你很容易把母语说得通顺流畅，却很容易把非母语说得不自然。
29474	I got fired from the company, but since I have...	虽然我被公司解雇了，但是我还有点存款，所以目前不用担心生计问题。
29475	If a person has not had a chance to acquire hi...	如果一個人在成人前沒有機會習得目標語言，他對該語言的認識達到母語者程度的機會是相當小的。

In [53]:
```
len(data)
```

Out[53]:
```
29476
```

判断有没有缺失值

In [54]:
```
data.isna().sum()
```

Out[54]:
```
English    0
Chinese    0
dtype: int64
```

第2步：切词

In [55]:
```
def tokenize(data):
    eng,cn=[],[]
    for x in data.English:
        eng.append(nltk.tokenize.word_tokenize(str(x).lower()))
    for y in data.Chinese:
```

```
        cn.append(list(jieba.cut(str(y))))
    return eng,cn
```

In [56]:
```
english,chinese=tokenize(data)
len(english),len(chinese)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.395 seconds.
Prefix dict has been built successfully.
```

Out[56]: (29476, 29476)

第3步:制作词典

In [57]:
```
class Vocab:
    def __init__(self,src,tgt):
        self.src=src
        self.tgt=tgt
        self.src_token2id={"PAD":0,"UNK":1,"BOS":2,"EOS":3}
        self.src_id2token={}
        self.tgt_token2id={"PAD":0,"UNK":1,"BOS":2,"EOS":3}
        self.tgt_id2token={}
        self.get_word2id()
        self.get_id2word()
    def get_word2id(self):
        src_merge=[word for seq in self.src for word in seq]
        temp1=Counter(src_merge)
        temp1 = sorted(temp1.items(), key = lambda x:x[1], reverse = True)
        for word, freq in temp1:
            id = len(self.src_token2id)
            self.src_token2id[word] = id

        tgt_merge=[word for seq in self.tgt for word in seq]
        temp2=Counter(tgt_merge)                                    # 统计词频
        temp2 = sorted(temp2.items(), key = lambda x:x[1], reverse = True) # 排序
        for word, freq in temp2:
            id = len(self.tgt_token2id)
            self.tgt_token2id[word] = id
    def get_id2word(self):
        self.src_id2token={value:key for (key,value) in self.src_token2id.items()}
        self.tgt_id2token={value:key for (key,value) in self.tgt_token2id.items()}
```

In [58]:
```
vocab=Vocab(english,chinese)
```

In [59]:
```
len(vocab.tgt_token2id)
```

Out[59]: 16259

第4步:字符转id

In [60]:
```
def word2id(seqs,vocab):
    temp=[]
    for seq in seqs:
        temp.append([vocab.get(word,vocab["UNK"]) for word in seq])
    return temp
```

In [61]:
```
src,tgt=word2id(english,vocab.src_token2id),word2id(chinese,vocab.tgt_token2id)
```

In [62]:
```
english[:3],src[:3]
```

Out[62]: ([['hi', '.'], ['hi', '.'], ['run', '.']], [[1414, 4], [1414, 4], [533, 4]])

In [63]:
```
chinese[:3],tgt[:3]
```

Out[63]:
```
([['嗨', '。'], ['你好', '。'], ['你', '用', '跑', '的', '。']],
 [[2734, 4], [1096, 4], [8, 106, 319, 6, 4]])
```

In [64]:
```
english_lens=[len(seq) for seq in english]
chinese_lens=[len(seq) for seq in chinese]
max(english_lens),max(chinese_lens)          # 查看最长句子,作为位置编码的长度
```

Out[64]: (36, 31)

第5步:构造Dataset和Dataloader

In [65]:
```
class MyDataset(torch.utils.data.Dataset):
    def __init__(self,src,tgt):
        self.src=src
        self.tgt=tgt
    def __len__(self):
        assert len(self.src)==len(self.tgt)
        return len(self.src)
    def __getitem__(self,idx):
        return self.src[idx],self.tgt[idx]
```

In [66]:
```
mydataset=MyDataset(src,tgt)
```

In [67]:
```
mydataset[0]
```

Out[67]: ([1414, 4], [2734, 4])

定义collate_fn

输入:[( ),( ),( ),( )],其中每个( )是形如上方mydataset[0]的一条数据

输出:((src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask),label)

输出的格式是根据runner中对dataloader的使用决定的,两者保持一致

In [68]:
```
def collate_fn(batch_data):
    def creat_key_padding_mask(L,seq_lens):
        if type(seq_lens) == list:
            seq_lens=torch.tensor(seq_lens)
        seq_lens=seq_lens.unsqueeze(-1)  # Bx1
        mask=torch.arange(L)<seq_lens    # BxL
        return ~mask
    def creat_attn_mask(L):
        mask=torch.arange(L).unsqueeze(-1)<torch.arange(L).unsqueeze(0)
        return mask
    src_L=0  # 记录最大长度
    tgt_L=0
    src_lens=[] # 记录实际长度
    tgt_lens=[]
    srcs,tgts,labels=[],[],[]
    for src,tgt in batch_data:
        label=tgt+[3]    # 标签尾部添加结束符
        tgt=[2]+tgt  # 解码器输入前添加开始符
        src_lens.append(len(src))
        tgt_lens.append(len(tgt))
        src_L=max(src_L,len(src))
        tgt_L=max(tgt_L,len(tgt))
        srcs.append(torch.tensor(src))
        tgts.append(torch.tensor(tgt))
        labels.append(torch.tensor(label))
    src_idx=nn.utils.rnn.pad_sequence(srcs,batch_first=True)
    tgt_idx=nn.utils.rnn.pad_sequence(tgts,batch_first=True)
    labels=nn.utils.rnn.pad_sequence(labels,batch_first=True)
    src_key_padding_mask=creat_key_padding_mask(src_L,src_lens)
    tgt_key_padding_mask=creat_key_padding_mask(tgt_L,tgt_lens)
    decoder_attn_mask=creat_attn_mask(tgt_L)
    return (src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask),labels
```

In [69]:
```
dataloader=torch.utils.data.DataLoader(mydataset,batch_size=4,collate_fn=collate_fn)
```

In [70]:
```
for inputs,label in dataloader:
    src_idx,tgt_idx,src_key_padding_mask,tgt_key_padding_mask,decoder_attn_mask=inputs
    print(tgt_idx)
    print(tgt_key_padding_mask)
    print(decoder_attn_mask)
    print(label)
    break
```

```
tensor([[   2, 2734,    4,    0,    0,    0],
        [   2, 1096,    4,    0,    0,    0],
        [   2,    8,  106,  319,    6,    4],
        [   2, 5202,   71,    0,    0,    0]])
tensor([[False, False, False,  True,  True,  True],
        [False, False, False,  True,  True,  True],
        [False, False, False, False, False, False],
        [False, False, False,  True,  True,  True]])
tensor([[False,  True,  True,  True,  True,  True],
        [False, False,  True,  True,  True,  True],
        [False, False, False,  True,  True,  True],
        [False, False, False, False,  True,  True],
        [False, False, False, False, False,  True],
        [False, False, False, False, False, False]])
tensor([[2734,    4,    3,    0,    0,    0],
        [1096,    4,    3,    0,    0,    0],
        [   8,  106,  319,    6,    4,    3],
        [5202,   71,    3,    0,    0,    0]])
```

第10节 实战-机器翻译

汇总上面所有层,并训练模型

Dataset与DataLoader

In [71]:
```
dataloader=torch.utils.data.DataLoader(mydataset,batch_size=180,collate_fn=collate_fn)
```

模型

In [72]:
```
# 基本组件实例化
src_wordembedding=WordEmbedding(len(vocab.src_token2id),96)
tgt_wordembedding=WordEmbedding(len(vocab.tgt_token2id)+3,96)  # 加上3个特殊字符unk,bos,eos
posembedding=PositionEmbedding(40,96)
encoder=EncoderBlock(d_model=96,fnn_mid_size=192,num_heads=8,normalized_shape=96)
decoder=DecoderBlock(d_model=96,fnn_mid_size=192,num_heads=8,normalized_shape=96)
encoderdecoder=EncoderDecoder(encoderblock=encoder,decoderblock=decoder,n1=6,n2=8)
classfier=Classifier(hidden_size=96,num_classes=len(vocab.tgt_token2id)+3)
```

In [73]:
```
# 组装Transformer
model=Transformer(src_wordembedding=src_wordembedding,tgt_wordembedding=tgt_wordembedding,positionembedding=posembedding,encoderdecoder=encoderdecoder,classifier=classfier).to(device)
```

损失函数、优化器、评估器

In [74]:
```
loss_fn=nn.CrossEntropyLoss(ignore_index=0,reduction="mean")
opt=torch.optim.Adam(model.parameters(),lr=0.002)
bleu=Bleu(n_size=3)
runner=Transformer_Runner(model=model,optimizer=opt,loss_fn=loss_fn,metric=bleu)
```

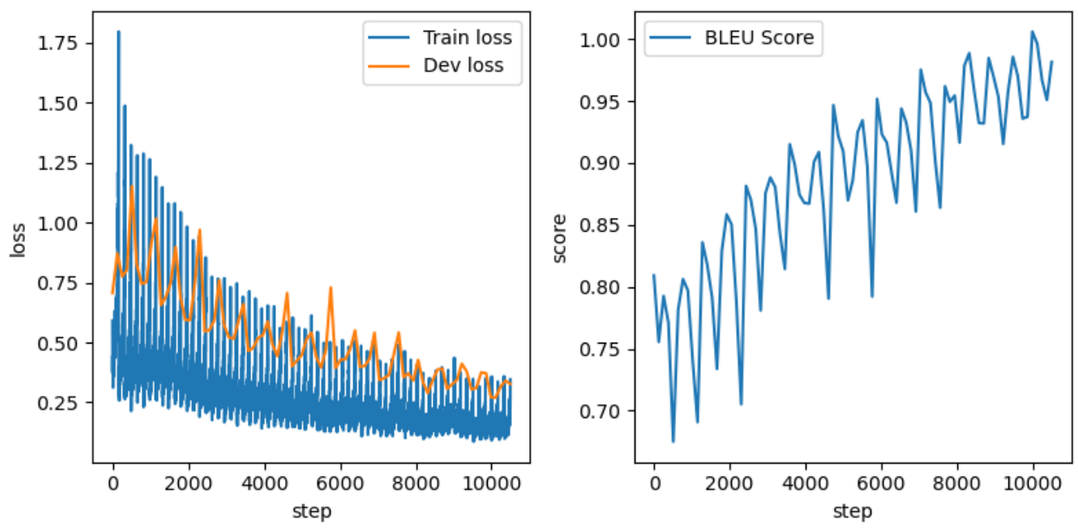
开始训练

In [75]:
```
runner.load(r"C:\Users\Administrator\Desktop\Open1507Lab\nndl-tutorial\step2432-score0.81273.params")
```

In [76]:
```
runner.train(num_epochs=64,train_loader=dataloader,dev_loader=dataloader,log_stride=128)
```

```
[Train]  epoch:0/64 step:0/10496 loss:0.5899
[Evaluate]  loss:0.7058 score:0.80910
Best model has been updated and saved!
[Train]  epoch:0/64 step:128/10496 loss:0.7159
[Evaluate]  loss:0.8714 score:0.75544
[Train]  epoch:1/64 step:256/10496 loss:0.4192
[Evaluate]  loss:0.7748 score:0.79251
[Train]  epoch:2/64 step:384/10496 loss:0.3663
[Evaluate]  loss:0.8025 score:0.77081
[Train]  epoch:3/64 step:512/10496 loss:0.3423
[Evaluate]  loss:1.1505 score:0.67492
[Train]  epoch:3/64 step:640/10496 loss:0.7899
[Evaluate]  loss:0.8260 score:0.78153
[Train]  epoch:4/64 step:768/10496 loss:0.4349
[Evaluate]  loss:0.7439 score:0.80615
[Train]  epoch:5/64 step:896/10496 loss:0.4033
[Evaluate]  loss:0.7520 score:0.79705
[Train]  epoch:6/64 step:1024/10496 loss:0.3875
[Evaluate]  loss:0.9096 score:0.73991
[Train]  epoch:7/64 step:1152/10496 loss:0.3122
[Evaluate]  loss:1.0164 score:0.69068
[Train]  epoch:7/64 step:1280/10496 loss:0.5182
[Evaluate]  loss:0.6553 score:0.83589
Best model has been updated and saved!
[Train]  epoch:8/64 step:1408/10496 loss:0.3685
[Evaluate]  loss:0.6893 score:0.81760
[Train]  epoch:9/64 step:1536/10496 loss:0.4224
[Evaluate]  loss:0.7473 score:0.79121
[Train]  epoch:10/64 step:1664/10496 loss:0.2578
[Evaluate]  loss:0.8979 score:0.73361
[Train]  epoch:10/64 step:1792/10496 loss:0.6610
[Evaluate]  loss:0.6914 score:0.82897
[Train]  epoch:11/64 step:1920/10496 loss:0.3702
[Evaluate]  loss:0.5938 score:0.85838
Best model has been updated and saved!
[Train]  epoch:12/64 step:2048/10496 loss:0.3587
[Evaluate]  loss:0.5932 score:0.85031
[Train]  epoch:13/64 step:2176/10496 loss:0.2502
[Evaluate]  loss:0.7503 score:0.78789
[Train]  epoch:14/64 step:2304/10496 loss:0.2819
[Evaluate]  loss:0.9689 score:0.70517
[Train]  epoch:14/64 step:2432/10496 loss:0.4039
[Evaluate]  loss:0.5459 score:0.88143
Best model has been updated and saved!
[Train]  epoch:15/64 step:2560/10496 loss:0.2774
[Evaluate]  loss:0.5520 score:0.86993
[Train]  epoch:16/64 step:2688/10496 loss:0.2869
[Evaluate]  loss:0.5928 score:0.84653
[Train]  epoch:17/64 step:2816/10496 loss:0.2873
[Evaluate]  loss:0.7584 score:0.78088
[Train]  epoch:17/64 step:2944/10496 loss:0.5712
[Evaluate]  loss:0.5735 score:0.87560
[Train]  epoch:18/64 step:3072/10496 loss:0.2878
[Evaluate]  loss:0.5210 score:0.88822
Best model has been updated and saved!
[Train]  epoch:19/64 step:3200/10496 loss:0.2574
[Evaluate]  loss:0.5156 score:0.88042
[Train]  epoch:20/64 step:3328/10496 loss:0.2646
[Evaluate]  loss:0.5909 score:0.84256
[Train]  epoch:21/64 step:3456/10496 loss:0.2047
[Evaluate]  loss:0.6592 score:0.81437
[Train]  epoch:21/64 step:3584/10496 loss:0.3575
[Evaluate]  loss:0.4643 score:0.91508
Best model has been updated and saved!
[Train]  epoch:22/64 step:3712/10496 loss:0.2483
[Evaluate]  loss:0.4775 score:0.89908
[Train]  epoch:23/64 step:3840/10496 loss:0.2602
[Evaluate]  loss:0.5176 score:0.87436
[Train]  epoch:24/64 step:3968/10496 loss:0.2195
[Evaluate]  loss:0.5306 score:0.86767
[Train]  epoch:24/64 step:4096/10496 loss:0.5580
[Evaluate]  loss:0.5880 score:0.86715
[Train]  epoch:25/64 step:4224/10496 loss:0.2619
[Evaluate]  loss:0.4903 score:0.90088
[Train]  epoch:26/64 step:4352/10496 loss:0.2146
[Evaluate]  loss:0.4428 score:0.90877
[Train]  epoch:27/64 step:4480/10496 loss:0.2473
[Evaluate]  loss:0.5462 score:0.86162
[Train]  epoch:28/64 step:4608/10496 loss:0.1516
[Evaluate]  loss:0.7055 score:0.79043
[Train]  epoch:28/64 step:4736/10496 loss:0.2899
[Evaluate]  loss:0.4008 score:0.94661
Best model has been updated and saved!
[Train]  epoch:29/64 step:4864/10496 loss:0.2628
[Evaluate]  loss:0.4235 score:0.92180
[Train]  epoch:30/64 step:4992/10496 loss:0.2405
[Evaluate]  loss:0.4476 score:0.90955
[Train]  epoch:31/64 step:5120/10496 loss:0.1632
[Evaluate]  loss:0.5127 score:0.86981
[Train]  epoch:32/64 step:5248/10496 loss:0.2250
[Evaluate]  loss:0.5412 score:0.88633
[Train]  epoch:32/64 step:5376/10496 loss:0.2367
[Evaluate]  loss:0.4389 score:0.92482
[Train]  epoch:33/64 step:5504/10496 loss:0.1836
[Evaluate]  loss:0.3957 score:0.93438
[Train]  epoch:34/64 step:5632/10496 loss:0.1567
[Evaluate]  loss:0.4699 score:0.89684
[Train]  epoch:35/64 step:5760/10496 loss:0.1841
[Evaluate]  loss:0.7286 score:0.79211
[Train]  epoch:35/64 step:5888/10496 loss:0.2837
[Evaluate]  loss:0.3935 score:0.95174
Best model has been updated and saved!
[Train]  epoch:36/64 step:6016/10496 loss:0.2227
[Evaluate]  loss:0.4282 score:0.92297
[Train]  epoch:37/64 step:6144/10496 loss:0.1900
[Evaluate]  loss:0.4299 score:0.91638
[Train]  epoch:38/64 step:6272/10496 loss:0.1915
[Evaluate]  loss:0.4719 score:0.89174
[Train]  epoch:39/64 step:6400/10496 loss:0.1539
[Evaluate]  loss:0.5502 score:0.86788
[Train]  epoch:39/64 step:6528/10496 loss:0.2313
[Evaluate]  loss:0.3987 score:0.94389
[Train]  epoch:40/64 step:6656/10496 loss:0.1971
[Evaluate]  loss:0.4018 score:0.93265
[Train]  epoch:41/64 step:6784/10496 loss:0.2377
[Evaluate]  loss:0.4328 score:0.91010
[Train]  epoch:42/64 step:6912/10496 loss:0.1452
[Evaluate]  loss:0.5405 score:0.86072
[Train]  epoch:42/64 step:7040/10496 loss:0.2815
[Evaluate]  loss:0.3425 score:0.97525
Best model has been updated and saved!
[Train]  epoch:43/64 step:7168/10496 loss:0.2063
[Evaluate]  loss:0.3510 score:0.95708
[Train]  epoch:44/64 step:7296/10496 loss:0.1630
[Evaluate]  loss:0.3648 score:0.94840
[Train]  epoch:45/64 step:7424/10496 loss:0.1825
[Evaluate]  loss:0.4517 score:0.90112
[Train]  epoch:46/64 step:7552/10496 loss:0.1902
[Evaluate]  loss:0.5410 score:0.86386
[Train]  epoch:46/64 step:7680/10496 loss:0.2648
[Evaluate]  loss:0.3551 score:0.96192
[Train]  epoch:47/64 step:7808/10496 loss:0.1973
[Evaluate]  loss:0.3715 score:0.94938
[Train]  epoch:48/64 step:7936/10496 loss:0.1812
[Evaluate]  loss:0.3412 score:0.95438
[Train]  epoch:49/64 step:8064/10496 loss:0.1428
[Evaluate]  loss:0.4271 score:0.91641
[Train]  epoch:49/64 step:8192/10496 loss:0.2623
[Evaluate]  loss:0.3340 score:0.97869
Best model has been updated and saved!
[Train]  epoch:50/64 step:8320/10496 loss:0.1491
[Evaluate]  loss:0.2892 score:0.98865
Best model has been updated and saved!
[Train]  epoch:51/64 step:8448/10496 loss:0.1533
[Evaluate]  loss:0.3418 score:0.95889
[Train]  epoch:52/64 step:8576/10496 loss:0.2000
[Evaluate]  loss:0.3872 score:0.93216
[Train]  epoch:53/64 step:8704/10496 loss:0.1885
[Evaluate]  loss:0.3904 score:0.93190
[Train]  epoch:53/64 step:8832/10496 loss:0.2152
[Evaluate]  loss:0.3067 score:0.98471
[Train]  epoch:54/64 step:8960/10496 loss:0.1782
[Evaluate]  loss:0.3259 score:0.96969
[Train]  epoch:55/64 step:9088/10496 loss:0.1812
[Evaluate]  loss:0.3432 score:0.95335
[Train]  epoch:56/64 step:9216/10496 loss:0.1758
[Evaluate]  loss:0.4113 score:0.91537
```

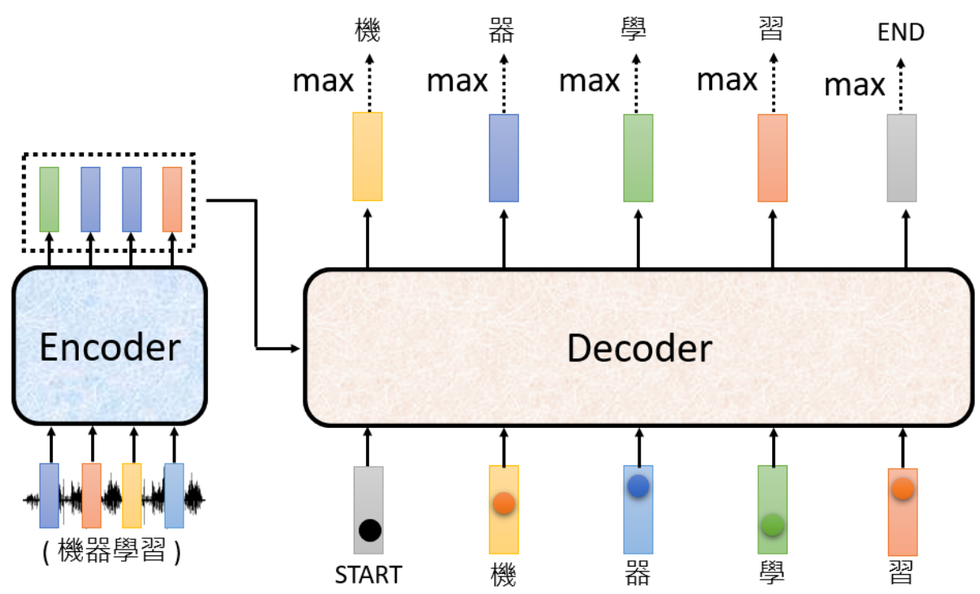
```
[Train]  epoch:56/64 step:9344/10496 loss:0.2825
[Evaluate]  loss:0.3796 score:0.95788
[Train]  epoch:57/64 step:9472/10496 loss:0.1583
[Evaluate]  loss:0.3044 score:0.98560
[Train]  epoch:58/64 step:9600/10496 loss:0.1539
[Evaluate]  loss:0.3105 score:0.97006
[Train]  epoch:59/64 step:9728/10496 loss:0.1527
[Evaluate]  loss:0.3752 score:0.93577
[Train]  epoch:60/64 step:9856/10496 loss:0.1157
[Evaluate]  loss:0.3718 score:0.93723
[Train]  epoch:60/64 step:9984/10496 loss:0.2026
[Evaluate]  loss:0.2717 score:1.00598
Best model has been updated and saved!
[Train]  epoch:61/64 step:10112/10496 loss:0.1678
[Evaluate]  loss:0.2706 score:0.99630
[Train]  epoch:62/64 step:10240/10496 loss:0.1682
[Evaluate]  loss:0.3160 score:0.96681
[Train]  epoch:63/64 step:10368/10496 loss:0.1302
[Evaluate]  loss:0.3406 score:0.95079
[Train]  epoch:63/64 step:10495/10496 loss:0.3458
[Evaluate]  loss:0.3274 score:0.98158
done!
```



第11节 预测

单独写成一个函数好了，就不封装到runner当中了，runner里面训练评估保存先搞熟悉，
其它东西暂时不熟就不往里加了，不然混在一起容易晕

预测函数设计思路简要



几个记号：
state=Encoder("机器学习")
Decoder记为概率函数P
预测步骤：
step1： 计算第一个概率分布P(word1|state,"start"),使用argmax P(word1|"start",state)求出word1
step2: 计算第二个概率分布P(word2|state,"start",word1),使用argmax P(word2|state,"start",word1)求出word2
step3: 计算第三个概率分布P(word3|state,"start",word1,word2),使用argmax P(word2|state,"start",word1,word2)求出word3
step4: 计算第四个概率分布P(word4|state,"start",word1,word2,word3),使用argmax P(word2|state,"start",word1,word2,word3)求出word4
step5: 计算第五个概率分布P(word5|state,"start",word1,word2,word3,word4),使用argmax P(word2|state,"start",word1,word2,word3,word4)求出word5
假如word5=="end",我们就不求概率分布了,把word1到word4串起来输出,作为翻译的结果

题外话1:
这个自注意力的输入是可变长的,可以吃一个,两个,三个,多个单词,这就表明它可以充当多个不同的概率函数P,
语言模型就是对多个P建模,实际上上面每个P你都能使用表格法对数据集进行统计建模(数数,count),或者使用
其他什么带参数的分布进行表示,再或者使用不同的神经网络进行表示,能理解这点就再好不过了
题外话2:
上面在预测阶段求的多个P,实际上在训练阶段通过decoder_attn_mask掩码机制可以一次性并行算出,并且和你单独一个一个算结果一致,如果不一致,
那么上面训练出来的模型将和后面我们预测计算的东西不一致,这样训练出来的东西毫无意义,可以自己去推导演算看看是不是这样
题外话3:
关于pad的掩码,就是为了保证和你单独计算一个句子的输出结果保持一致,其实也可以直接把数据集改成带pad的,然后干脆不做掩码了,
就当pad也是正常的字符,统一训练,这样训练出来后,很多句子的预测也会带pad,在输出的时候删除即可,这个方法虽然简单粗暴,但是需要
所有句子pad成统一的长度,如果句子长短差异太大,短的句子填充太多pad,会不会对句子的表征产生不良影响不得而知

开始一步一步进行预测

确定输入和start

In [111…
```
origin="They drink coke."
origin=nltk.tokenize.word_tokenize(origin.lower())
origin
```
Out[111]:
```
['they', 'drink', 'coke', '.']
```

In [112…
```
src=torch.tensor([vocab.src_token2id.get(word,1) for word in origin]).unsqueeze(0)
src
```
Out[112]:
```
tensor([[  56,  251, 2830,    4]])
```

In [113…
```
start=torch.tensor(vocab.tgt_token2id["BOS"]).unsqueeze(0).unsqueeze(0)
start
```
Out[113]:
```
tensor([[2]])
```

求第一个单词

In [114…
```
output=model(src,start,None,None,None)
```

In [115…
```
pred=F.softmax(output,dim=-1)
```

In [116…
```
word1=torch.argmax(pred,dim=-1)
word1
```
Out[116]:
```
tensor([[57]])
```

In [117…
```
print(vocab.tgt_id2token[word1.item()])
```

他們

求第二个单词

注意,输入两个单词,第一个单词每次经过自注意力层需要掩码, 保证第一个单词的编码与训练的时候每层的编码一致,这样其他单词与其交互计算出来的自身的编码才是正确的

In [118…
```
tgt=torch.concat([start,word1],dim=1)
tgt
```

Out[118]:
```
tensor([[ 2, 57]])
```

In [119…
```
attn_mask=creat_attn_mask(2)
```

In [120…
```
output=model(src,tgt,None,None,attn_mask)
pred=F.softmax(output,dim=-1)
words=torch.argmax(pred,dim=-1)
words
```

Out[120]:
```
tensor([[ 57, 273]])
```

In [121…
```
word2=words[0,1]
word2
```

Out[121]:
```
tensor(273)
```

In [122…
```
print(vocab.tgt_id2token[word2.item()])
```

喝

求第三个单词

In [123…
```
tgt=torch.concat([start,words],dim=1)
attn_mask=creat_attn_mask(3)
output=model(src,tgt,None,None,attn_mask)
pred=F.softmax(output,dim=-1)
words=torch.argmax(pred,dim=-1)
words
```

Out[123]:
```
tensor([[  57,  273, 5353]])
```

In [124…
```
word3=words[0,2]
print(vocab.tgt_id2token[word3.item()])
```

可口

求第四个单词

In [125…
```
tgt=torch.concat([start,words],dim=1)
attn_mask=creat_attn_mask(4)
output=model(src,tgt,None,None,attn_mask)
pred=F.softmax(output,dim=-1)
words=torch.argmax(pred,dim=-1)
word4=words[0,3]
print(vocab.tgt_id2token[word4.item()])
```

可樂

求第五个单词

In [126…
```
tgt=torch.concat([start,words],dim=1)
attn_mask=creat_attn_mask(5)
output=model(src,tgt,None,None,attn_mask)
pred=F.softmax(output,dim=-1)
words=torch.argmax(pred,dim=-1)
word5=words[0,4]
print(vocab.tgt_id2token[word5.item()])
```

。

求终止符

In [128…
```
tgt=torch.concat([start,words],dim=1)
attn_mask=creat_attn_mask(6)
output=model(src,tgt,None,None,attn_mask)
pred=F.softmax(output,dim=-1)
words=torch.argmax(pred,dim=-1)
end=words[0,5]
print(vocab.tgt_id2token[end.item()])
```

EOS

根据上面的过程设计预测函数

输入:一段英文字符串
输出:一段中文字符串

In [148…
```
def translator(seq):
    origin=nltk.tokenize.word_tokenize(seq.lower())
    src=torch.tensor([vocab.src_token2id.get(word,1) for word in origin]).unsqueeze(0)
    start=torch.tensor(vocab.tgt_token2id["BOS"]).unsqueeze(0).unsqueeze(0)
    words=torch.Tensor([[]]).long()
    end=False
    L=1
    seq_out=[]
    while not end:
        tgt=torch.concat([start,words],dim=1)
        attn_mask=creat_attn_mask(L)
        output=model(src,tgt,None,None,attn_mask)
        pred=F.softmax(output,dim=-1)
        words=torch.argmax(pred,dim=-1)
        word=words[0,L-1]
        L+=1
        if word.item()==3:
            break
        seq_out.append(vocab.tgt_id2token[word.item()])
    print("".join(seq_out))
```

In [149…
```
translator("They drink coke.")
```

他們喝可口可樂。

In [150…
```
translator("I can't tell a frog from a toad.")
```

我无法区分青蛙和蟾蜍。