[详解PyTorch中的ModuleList和Sequential - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/75206669)

[【nn.Parameter()】生成和为什么要初始化\_nn.module中有自动初始化吗\_panbaoran913的博客-CSDN博客](https://blog.csdn.net/panbaoran913/article/details/126070317)

torch.nn.Parameter是继承自torch.Tensor的子类，其主要作用是作为nn.Module中的可训练参数使用。它与torch.Tensor的区别就是nn.Parameter会自动被认为是module的可训练参数，即加入到parameter()这个迭代器中去；而module中非nn.Parameter()的普通tensor是不在parameter中的。

* 调用module时传入参数与module类定义的forward参数一致

[Pytorch nn.Module和forward的参数关系\_moudule中forward的参数怎么取得\_小哈蒙德的博客-CSDN博客](https://blog.csdn.net/qq_38109282/article/details/117373550)

[pytorch masked\_fill方法理解 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/151783950)

[torch.unsqueeze() 和 torch.squeeze() - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/86763381)

返回一个新的张量，对输入的既定位置插入维度 1

[Pytorch中contiguous()函数理解\_.contiguous()\_清晨的光明的博客-CSDN博客](https://blog.csdn.net/kdongyi/article/details/108180250)

**不改变tensor的内容本身**，而只是**重新定义下标与元素的对应关系**

当**调用contiguous()时**，**会强制拷贝一份tensor**，让它的布局和从头创建的一模一样，**但是两个tensor完全没有联系**。

register\_buffer 这个方法的作用在于定义一组参数，特殊点在于训练模型中register\_buffer 定义的参数不能被更新。

[torch中register\_buffer - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/576260927)

model = EncoderDecoder(

        Encoder(EncoderLayer(d\_model, c(attn), c(ff), dropout), N),

        Decoder(DecoderLayer(d\_model, c(attn), c(attn), c(ff), dropout), N),

        nn.Sequential(Embeddings(d\_model, src\_vocab), c(position)),

        nn.Sequential(Embeddings(d\_model, tgt\_vocab), c(position)),

        Generator(d\_model, tgt\_vocab)

    )

label smoothing: regularization

It involves adding a small amount of uncertainty or noise to the labels used for training. In the case of the Transformer model, label smoothing is often applied to the target labels in sequence-to-sequence tasks, such as machine translation or language generation.

Instead of assigning a probability of 1 to the true label and 0 to all other labels, label smoothing assigns a probability that is slightly lower than 1 to the true label and distributes the remaining probability mass among other labels.

The effect of label smoothing is that the model is encouraged to be less confident in its predictions and is incentivized to explore alternative possibilities. This can prevent the model from becoming overly specialized to the training data and improve its generalization to unseen examples.

[pytorch系列-----1 python class 中 的\_\_call\_\_方法\_pytorchcall方法\_墨氲的博客-CSDN博客](https://blog.csdn.net/dss_dssssd/article/details/83750838)

python类中的\_\_call\_\_\_: 使得类对象具有类似函数的功能。可以按照调用函数的方式调用。

[Pytorch中.detach()与.data的用法 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/410199046)

Tensor.data和Tensor.detach()一样， 都会返回一个新的Tensor， 这个Tensor和原来的Tensor共享内存空间，一个改变，另一个也会随着改变，且都会设置新的Tensor的requires\_grad属性为False。这两个方法只取出原来Tensor的tensor数据， 丢弃了grad、grad\_fn等额外的信息。区别在于Tensor.data不能被autograd追踪到，如果你修改了Tensor.data返回的新Tensor，原来的Tensor也会改变， 但是这时候的微分并没有被追踪到，那么当你执行loss.backward()的时候并不会报错，但是求的梯度就是错误的！因此， 如果你使用了Tensor.data，那么切记一定不要随便修改返回的新Tensor的值。如果你使用的是Tensor.detach()方法，当你修改他的返回值并进行求导操作，会报错。 因此，Tensor.detach()是安全的。

[torchtext.vocab — Torchtext 0.15.0 documentation (pytorch.org)](https://pytorch.org/text/stable/vocab.html#build-vocab-from-iterator)

build\_vocab\_from\_iterator

**>>>** *#generating vocab from text file*

**>>> import** **io**

**>>> from** **torchtext.vocab** **import** **build\_vocab\_from\_iterator**

**>>> def** **yield\_tokens(file\_path):**

**>>>**  **with** **io.open(file\_path,** **encoding** **=** 'utf-8'**)** **as** **f:**

**>>>**  **for** **line** **in** **f:**

**>>>**  **yield** **line.strip().split()**

**>>> vocab** **=** **build\_vocab\_from\_iterator(yield\_tokens(file\_path),** **specials=[**"<unk>"**])**