VIT-v1v

CSDN上一篇关于介绍Transformer的文章，比较详细，而且数据比较简单

【https://zhilengnuan.blog.csdn.net/article/details/121101749?fromshare=blogdetail&sharetype=blogdetail&sharerId=121101749&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【1】归一化的层

```
self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()
```

关于Normalization

理论：【https://blog.csdn.net/vict_wang/article/details/88075861?fromshare=blogdetail&sharetype=blogdetail&sharerId=88075861&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

附有代码：【https://chukai.blog.csdn.net/article/details/108572992?fromshare=blogdetail&sharetype=blogdetail&sharerId=108572992&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【2】有关Q,K,V

```
self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
```

【https://blog.csdn.net/weixin_45303602/article/details/134188049?fromshare=blogdetail&sharetype=blogdetail&sharerId=134188049&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【偏置】

总的来说，偏置项的作用是调整神经元的激活函数输出，使得神经网络能够更好地拟合输入数据的分布，**提高模型的泛化能力。**在网络的初始阶段，偏置可以初始化为小的随机值，然后通过反向传播算法进行训练，以使网络能够适应输入数据并**调整偏置的值**，以最小化损失函数。

Q：在实现多头注意力机制时为什么不用自己定义的DropPath?

A：在神经网络模型中，选择使用 `nn.Dropout` 还是自己定义的 `drop_path` 函数取决于具体的应用场景和模型架构的需求。

首先，`nn.Dropout` 是 PyTorch 提供的一个标准模块，用于在训练过程中随机丢弃输入张量的一部分元素。它通常用于全连接层（线性层）或卷积层的输出，以减少模型的过拟合。`nn.Dropout` 的工作方式是简单的，它接受一个丢弃概率作为参数，并根据这个概率将输入张量中的元素置零。

然而，`drop_path` 函数（如之前定义的）通常用于更具体的场景，比如在 Transformer 模型中的注意力机制之后。`drop_path` 的目的可能是为了随机丢弃一些注意力头的输出路径，以增加模型的泛化能力。这种路径丢弃机制与标准的 Dropout 有所不同，因为它通常是在特定的网络层（如多头注意力层）之后应用的，并且可能涉及到对保留路径的缩放以保持输出的期望值不变。

具体：

1. **标准性与兼容性**：`nn.Dropout` 是 PyTorch 框架中的一个标准模块，它的行为是已知的、可预测的，并且与框架中的其他模块兼容。使用标准模块有助于确保模型的稳定性和可维护性。
2. **简化代码**：如果 `nn.Dropout` 已经满足了你的需求（即在注意力机制之后应用丢弃），那么就没有必要自己定义一个类似的函数。使用标准模块可以减少代码量，并降低出错的风险。
3. **可配置性**：`nn.Dropout` 提供了简单的接口来配置丢弃概率，这使得在训练过程中调整丢弃率变得容易。相比之下，自己定义的 `drop_path` 函数可能需要额外的参数或逻辑来处理不同的场景。
4. **性能优化**：PyTorch 的标准模块通常经过了优化，可以在不同的硬件上高效地运行。使用这些模块有助于确保你的模型在训练和推理过程中具有良好的性能。
5. **社区支持**：使用标准模块意味着你可以利用 PyTorch 社区提供的广泛支持和资源。如果遇到问题，你可以更容易地找到解决方案或获得帮助。

总之，选择使用 `nn.Dropout` 还是 `drop_path` 取决于你的具体需求。如果 `nn.Dropout` 已经满足了你的需求，并且你希望保持代码的简洁性和可维护性，那么使用它可能是更好的选择。如果你需要实现更复杂的丢弃逻辑（如路径丢弃），那么自己定义一个函数可能是必要的。

【3】qkv操作

```
qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, C //
self.num_heads).permute(2, 0, 3, 1, 4)
```

```
# qkv(): -> [batch_size, num_patches + 1, 3 * total_embed_dim]
# reshape: -> [batch_size, num_patches + 1, 3, num_heads, embed_dim_per_head]
# permute: -> [3, batch_size, num_heads, num_patches + 1, embed_dim_per_head]
```

reshape重塑操作：

reshape操作不改变张量的数据，只是改变其形状。

在这个例子中，我们将qkv的输出重塑为一个五维张量，其中：
batch_size保持不变，表示批次大小。
num_patches + 1也保持不变，表示输入数据中的块（或序列中的元素）数量加一。
新增了一个维度3，用于区分查询、键和值。
num_heads是注意力头的数量，是自注意力机制中的一个关键参数。
embed_dim_per_head是每个头的嵌入维度，它通常是total_embed_dim / num_heads的整数倍

（在大多数情况下，total_embed_dim会被设计成num_heads * embed_dim_per_head的形式，以确保整除）。

permute置换操作：

用来重新排列张量的维度的

自注意力机制中，我们通常希望将查询（query）、键（key）和值（value）分开处理，并且希望每个注意力头（attention head）能够独立地处理输入数据。通过置换操作，我们可以将查询、键和值分别放在张量的第一个维度上，这样后续操作就可以很方便地针对每个头、每个块（或序列元素）进行处理。

【4】全连接层

解释+代码：【https://blog.csdn.net/fydw_715/article/details/141867562?fromshare=blogdetail&sharetype=blogdetail&sharerId=141867562&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【5】前向传播和反向传播过程

【https://blog.csdn.net/m0_51200050/article/details/140014610?fromshare=blogdetail&sharetype=blogdetail&sharerId=140014610&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【6】蒸馏token

DeiT 是一个全 Transformer 的架构。其核心是提出了**针对 ViT 的教师-学生蒸馏训练策略**，并提出了 token-based distillation 方法，使得 Transformer 在视觉领域训练得又快又好。

知识蒸馏介绍

Knowledge Distillation（KD）最初被 Hinton 提出 “Distilling the Knowledge in a Neural Network”，与 Label smoothing 动机类似，但是 KD 生成 soft label 的方式是通过教师网络得到的。

KD 可以视为将教师网络学到的信息压缩到学生网络中。还有一些工作 “Circumventing outlier of autoaugment with knowledge distillation” 则将 KD 视为数据增强方法的一种。

虽然在一般情况下，我们不会去区分训练和部署使用的模型，但是训练和部署之间存在着一定的不一致性。在训练过程中，我们需要使用复杂的模型，大量的计算资源，以便从非常大、高度冗余的数据集中提取出信息。在实验中，效果最好的模型往往规模很大，甚至由多个模型集成得到。而大模型不方便部署到服务中去，常见的瓶颈如下:

推理速度和性能慢；

对部署资源要求高(内存，显存等)；

在部署时，对延迟以及计算资源都有着严格的限制。

因此，模型压缩（在保证性能的前提下减少模型的参数量）成为了一个重要的问题，而**“模型蒸馏”属于模型压缩的一种方法。**

理论原理

知识蒸馏使用的是 Teacher—Student 模型，其中 Teacher 是“知识”的输出者，Student 是“知识”的接受者。知识蒸馏的过程分为2个阶段:

原始模型训练: 训练 “Teacher模型”, 简称为**Net-T**，它的特点是**模型相对复杂**，也可以由多个分别训练的模型集成而成。我们对"Teacher模型"不作任何关于模型架构、参数量、是否集成方面的限制，唯一的要求就是，对于输入X, 其都能输出Y，其中Y经过softmax的映射，输出值对应相应类别的概率值。

精简模型训练: 训练"Student模型", 简称为Net-S，它是**参数量较小**、**模型结构相对简单**的单模型。同样的，对于输入X，其都能输出Y，Y经过softmax映射后同样能输出对应相应类别的概率值。

知识蒸馏时，由于已经有了一个泛化能力较强的Net-T，我们在利用Net-T来蒸馏训练Net-S时，可以**直接让Net-S去学习Net-T的泛化能力。**

Distillation Token 和 ViT 中的 class token 一起加入 Transformer 中，和class token 一样通过 self-attention 与其它的 embedding 一起计算，并且在最后一层之后由网络输出。

而 Distillation Token 对应的这个输出的目标函数就是蒸馏损失。Distillation Token 允许模型从教师网络的输出中学习，就像在常规的蒸馏中一样，同时也作为一种对class token的补充。

【7】有关token

token就是指单词或者语句

【https://blog.csdn.net/Soonki/article/details/140439403?fromshare=blogdetail&sharetype=blogdetail&sharerId=140439403&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【8】截断正态分布

截断分布是指，限制变量xx 取值范围(scope)的一种分布。例如，限制x取值在0到50之间，即{0<x<50}。

截断了的正态分布仍然保留着正态分布的许多特征，比如它的均值、方差、标准差等。**截断的影响主要表现在分布的尾部，即截断的区间之外。截断会使得分布在截断区间之外的概率变小，而在截断区间内的概率变大。**

【https://mapengsen.blog.csdn.net/article/details/120622761?fromshare=blogdetail&sharetype=blogdetail&sharerId=120622761&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【9】class token（类别令牌）

在VIT（Vision Transformer）中，class token（类别令牌）是一种特殊的**位置编码**，它被添加到图像的嵌入表示中，并且在训练过程中与图像的标签相关联。这个类别令牌的作用是**为模型提供关于整个图像类别的全局信息**，从而帮助模型学习对图像内容进行分类的表示。

作用：

全局信息集成: 类别令牌允许模型**在推理时通过整体图像的类别信息进行分类决策。**它捕捉了整个图像的语义内容，而不是仅仅依赖于图像中各个局部区域的特征。

联合训练: 类别令牌在训练过程中**与图像的标签进行联合**，这样模型可以学习**将全局特征与标签联系起来的有效表示。**这种方法有助于提高模型在分类任务中的性能。

预测过程：

在**预测阶段**，ViT模型使用类别令牌来**预测图像的类别。**具体步骤如下：

提取特征: 首先，ViT模型将输入的图像**分成若干个图像块**，并对每个图像块进行线性变换以获得初始的图像块表示。

加入类别令牌: 在初始的图像块表示中，类别令牌**被添加为一个额外的向量。**这个向量通常与模型的其他位置编码向量具有相同的维度。

Transformer编码: 将加入类别令牌的图像块表示作为输入，通过Transformer编码器进行多层次的自注意力机制和前馈网络操作，以学习图像的语义表示。

分类预测: 在Transformer的最后一个输出层之后，通常会**接一个全连接层或者类似的结构**，将最后一个位置的特征向量（通常是类别令牌的特征向量）映射到预测类别的空间。这个过程可以理解为一个简单的分类器，它基于全局的图像表示进行分类决策。

通过这种方式，类别令牌在ViT模型中发挥了关键作用，帮助模型有效地处理图像分类任务，并在推理时结合全局信息进行准确的预测。

【10】词嵌入与位置嵌入

"词嵌入"（Word Embedding）用于表示将词汇表中的**每个单词映射到一个高维向量空间中的技术。**而**"位置嵌入"**则是用来处理**序列数据中元素位置信息的嵌入技术。**

词嵌入（Word Embedding）

想象你有一个巨大的词库，里面包含了所有可能用到的单词，比如“猫”、“狗”、“跑”、“吃”等。词嵌入技术就是将这些单词统一转换成一个固定长度的向量（比如300维）。这个过程就像是给每个单词分配了一个独特的“身份证号码”，但这个“号码”不仅仅是一个简单的数字，而是一个包含了单词语义信息的向量。例如，“猫”和“狗”在向量空间中的位置可能比较接近，因为它们都是动物；而“跑”和“吃”则可能距离较远，因为它们的动作性质不同。

位置嵌入（Positional Embedding）

在自然语言处理中，除了单词本身的意义外，单词在句子中的位置也非常重要。位置嵌入就是用来编码这种位置信息的。假设你有一个句子：“猫吃鱼”，并且你已经将每个单词转换成了词嵌入向量。现在，你还需要为每个单词添加一个位置嵌入向量，以表示它们在句子中的位置。比如，“猫”是第一个词，所以它有一个表示第一个位置的位置嵌入；“吃”是第二个词，有第二个位置的位置嵌入，依此类推。

将这些位置嵌入向量与对应的词嵌入向量相加（或按其他方式组合），就得到了每个单词的最终表示，这个表示既包含了单词的语义信息，也包含了单词在句子中的位置信息。

【https://blog.csdn.net/YHKKun/article/details/137089868?fromshare=blogdetail&sharetype=blogdetail&sharerId=137089868&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】

【11】正则化

正则化就是说给损失函数加上一些限制，通过这种规则去规范他们再接下来的循环迭代中，不要自我膨胀。

【https://songjian.blog.csdn.net/article/details/104891561?fromshare=blogdetail&sharetype=blogdetail&sharerId=104891561&sharerefer=PC&sharesource=2301_80864377&sharefrom=from_link】