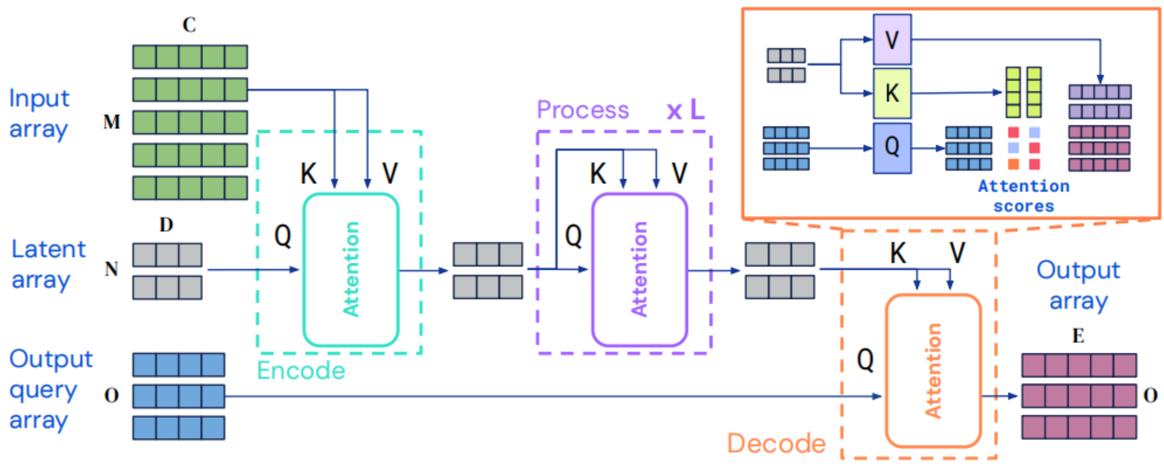
perceiver

perceiverIO

PerceiverIO就是Perceiver的进阶版本，其比在原有的编码器结构上做出了一定的调整，最大的变化是多了一个由cross attention构成的Perceiver结构用于解码编码器的语言信息。其结构如下图：



为了更具体地帮助理解输入数据经过PerceiverIO形状是如何改变的，接下来给出一段矩阵公式推导。在开始之前，先约定好数据的形状，符号分别如下：

$$\text{Input array}:\mathbf{X}=\begin{pmatrix}\mathbf{x}_1^T\\ \vdots\\ \mathbf{x}_k^T\end{pmatrix}\ \text{Latent array}:\mathbf{Z}=\begin{pmatrix}\mathbf{z}_1^T\\ \vdots\\ \mathbf{z}_m^T\end{pmatrix}\ \text{Output query array}:\mathbf{Z}'=\begin{pmatrix}\mathbf{z'_1}^T\\ \vdots\\ \mathbf{z'_n}^T\end{pmatrix}$$

$$\mathbf{x}_i\in\mathbb{R}^{kv_{dim}\times 1},\mathbf{z}_i\in\mathbb{R}^{q_{dim}\times 1},\mathbf{z'_i}\in\mathbb{R}^{kv_{dim}\times 1}$$
$$\mathbf{W}^Q\in\mathbb{R}^{q_{dim}\times qk_{channels}},\mathbf{W}^K\in\mathbb{R}^{kv_{dim}\times qk_{channels}},\mathbf{W}^V\in\mathbb{R}^{kv_{dim}\times v_{channels}}$$

下标dim对应了矩阵的行，下标为$channels$对应矩阵的列。因为k,v实际上是同一个东西,所以用kv_{dim}表示$\mathbf{W}^Q$的行维度,而$query$的维度和k,v其实没有关系,所以用q_{dim}表示$\mathbf{W}^Q$的列维度.由于QK^T计算，所以需要保证$\mathbf{W}^Q$和$\mathbf{W}^K$的列数相同，所以用二者的列数都记作$qk_{channels}$，最后$\mathbf{V}$的维度没有显示地要求，所以$\mathbf{W}^V$的列数用$v_{channels}$表示。

step1.输入矩阵和隐状态矩阵Latent array通过cross attention进行语义融合。输入$\mathbf{X}$经过三个$\mathbf{W^Q},\mathbf{W^K},\mathbf{W^V}$投影矩阵得到$Q,K,V$:

$$Q=\mathbf{ZW^Q}\in\mathbb{R}^{m\times qk_{channels}}$$
$$K=\mathbf{XW^K}\in\mathbb{R}^{k\times qk_{channels}}$$
$$V=\mathbf{XW^V}\in\mathbb{R}^{k\times v_{channels}}$$
$$QK^T=\mathbf{ZW^Q{W^K}^TX}\in\mathbb{R}^{m\times k}$$

$$\text{cross att output}=\mathrm{softmax}(\mathrm{QK^T}/\sqrt{\mathrm{d}})\mathrm{V}\in\mathbb{R}^{\mathrm{m\times v_{channels}}}(\text{此处省略multi-head操作})$$

从最终输出的结果可以看到，Cross attention的输出在seqlen(time index)和Latent array(Query)保持一致，在hidden dim上会和$v_{channels}$保持一致($v_{channels}$可以自己设置)。

step2.而接下来的自注意力模块的Q,K,V都来自Cross attention的输出,形状也不会改变.所以整个Encoder 模块的输出的形状就是$\text{Output}_{Encoder}\in\mathbb{R}^{m\times v_{channels}}$.

我们来看一下源码的具体实现:(transformers/models/perceiver/modeling_perceiver.py) 对于最开始的Cross attention,初始化时要指定q_{dim}, kv_{dim}和$qk_{channels}$, $v_{channels}$.而PerceiverLayer类的实现最终会定位到PerceiverSeflAttention(transformers/models/perceiver/modeling_perceiver.py).对应的代码是:

```
class PerceiverSelfAttention(nn.Module):
    """Multi-headed {cross, self}-attention. Can be used both in the encoder as
well as in the decoder."""
    def __init__(
        self,
        config,
        is_cross_attention=False,
        qk_channels=None,
        v_channels=None,
        num_heads=1,
        q_dim=None,
        kv_dim=None,
    ):
        super().__init__()
        self.num_heads = num_heads
        # Q and K must have the same number of channels.
        # Default to preserving Q's input's shape.
        if qk_channels is None:
            qk_channels = q_dim
        # V's num_channels determines the shape of the output of QKV-attention.
        # Default to the same number of channels used in the key-query operation.
        if v_channels is None:
            v_channels = qk_channels
        if qk_channels % num_heads != 0:
            raise ValueError(f"qk_channels ({qk_channels}) must be divisible by
num_heads ({num_heads}).")
        if v_channels % num_heads != 0:
            raise ValueError(f"v_channels ({v_channels}) must be divisible by
num_heads ({num_heads}).")

        self.qk_channels = qk_channels
        self.v_channels = v_channels
        self.qk_channels_per_head = self.qk_channels // num_heads
        self.v_channels_per_head = self.v_channels // num_heads
        #print("Perceiver Self attention 要求: layernorm1 q_dim:{}".format(q_dim))
        # Layer normalization
        self.layernorm1 = nn.LayerNorm(q_dim)
        #print("Perceiver Self attention 要求: layernorm2 KV_dim:
{}".format(kv_dim))
        self.layernorm2 = nn.LayerNorm(kv_dim) if is_cross_attention else
nn.Identity()

        # Projection matrices
        self.query = nn.Linear(q_dim, qk_channels) #[q_dim,qk_channels]
        self.key = nn.Linear(kv_dim, qk_channels) #[kv_dim,qk_channels]
        self.value = nn.Linear(kv_dim, v_channels) #[kv_dim,v_channels]

        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
```

```
    def transpose_for_scores(self, x, channels_per_head):
        new_x_shape = x.size()[:-1] + (self.num_heads, channels_per_head)
        x = x.view(*new_x_shape)
        return x.permute(0, 2, 1, 3)

    def forward(
        self,
        hidden_states: torch.Tensor,
        attention_mask: Optional[torch.FloatTensor] = None,
        head_mask: Optional[torch.FloatTensor] = None,
        inputs: Optional[torch.FloatTensor] = None,
        inputs_mask: Optional[torch.FloatTensor] = None,
        output_attentions: Optional[bool] = False,
    ) -> Tuple[torch.Tensor]:
        #print("进入Perceiver Self Attention 输入形状：{}".format(inputs.shape))
        hidden_states = self.layernorm1(hidden_states)  #hiddent state 就对应于 Latent array #[Batch,m,q_dim]
        inputs = self.layernorm2(inputs) #shape [Batch,k,kv_dim]
        # Project queries, keys and values to a common feature dimension. If this is instantiated as a cross-attention module,
        # the keys and values come from the inputs; the attention mask needs to be such that the inputs's non-relevant tokens are not attended to.
        is_cross_attention = inputs is not None
        queries = self.query(hidden_states)

        if is_cross_attention:
            '''交叉注意力机制'''
            keys = self.key(inputs) #[Batch,k,qk_channels]
            values = self.value(inputs) #[Batch,k, v_channels]
            attention_mask = inputs_mask  #[Batch,max_seqlen]
        else:
            keys = self.key(hidden_states)
            values = self.value(hidden_states)

        # Reshape channels for multi-head attention.
        # We reshape from (batch_size, time, channels) to (batch_size, num_heads, time, channels per head)
        queries = self.transpose_for_scores(queries, self.qk_channels_per_head)
        keys = self.transpose_for_scores(keys, self.qk_channels_per_head)
        values = self.transpose_for_scores(values, self.v_channels_per_head)

        # Take the dot product between the queries and keys to get the raw attention scores.
        attention_scores = torch.matmul(queries, keys.transpose(-1, -2))

        batch_size, num_heads, seq_len, q_head_dim = queries.shape
        _, _, _, v_head_dim = values.shape
        hiddens = self.num_heads * v_head_dim

        attention_scores = attention_scores / math.sqrt(q_head_dim)  #[Batch,q index len, kv index len]

        if attention_mask is not None:
            # Apply the attention mask (precomputed for all layers in PerceiverModel forward() function)
            attention_scores = attention_scores + attention_mask
```

```
        # Normalize the attention scores to probabilities.
        attention_probs = nn.Softmax(dim=-1)(attention_scores)

        # This is actually dropping out entire tokens to attend to, which might
        # seem a bit unusual, but is taken from the original Transformer paper.
        attention_probs = self.dropout(attention_probs)

        # Mask heads if we want to
        if head_mask is not None:
            attention_probs = attention_probs * head_mask

        context_layer = torch.matmul(attention_probs, values) #attention矩阵乘以
values [Batch,num_heads,q index len,v_channels] 一般情况下v_channels = q_dim

        context_layer = context_layer.permute(0, 2, 1, 3).contiguous() #[Batch,q
index len,num_heads,v_channels] 一般情况下v_channels = q_dim
        new_context_layer_shape = context_layer.size()[:-2] + (hiddens,)
        context_layer = context_layer.view(*new_context_layer_shape)    #[Batch,q
index len,num_heads*v_channels_per_head]

        outputs = (context_layer, attention_probs) if output_attentions else
(context_layer,)
```

step3.我们不妨来进行模型推理验证: 可以看到 输入$\mathbf{X}$形状是$[16, 175, 768]$,Latent array形状是$[16, 256, 512]$,cross attention 和Encoder最终的输出形状都是$[16, 256, 512]$.注意:实际上$v_{channels}$得和q_{dim}是保持一致的，虽然表明上没有显示地说明，但是因为残差链接的原因$Q +$ cross att output操作需要二者维度一直。所以Encoder最终的输出形状和Latent array是一致的。

step4.同样,对于Decoder而言,其输出在交叉注意力机制的影响下会和一致(具体流程和Encoder中一样,故次省略).所以,如果是用于Classification任务,那么Output Lantent array的形状就应该是:$[\text{Batch}, 1, \text{hidden dim}]$.这样Decoder的输出就会变成$[\text{Batch}, 1, \text{hidden dim}]$,squeeze掉第1维以后再接上用于分类的线性得到最终的输出形状就是$[\text{Batch}, \text{num classes}]$。

LUNA