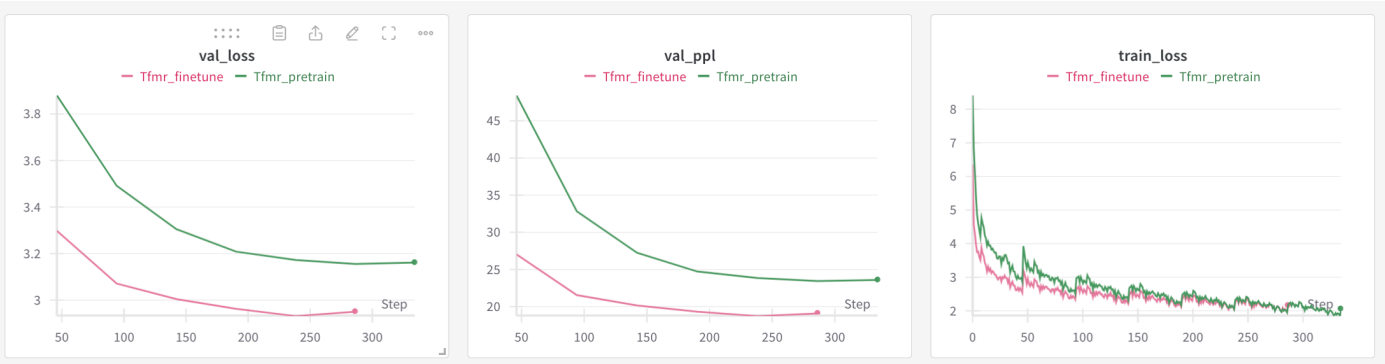
ANN HW3 Transformer

王浩然 计23 2022010229

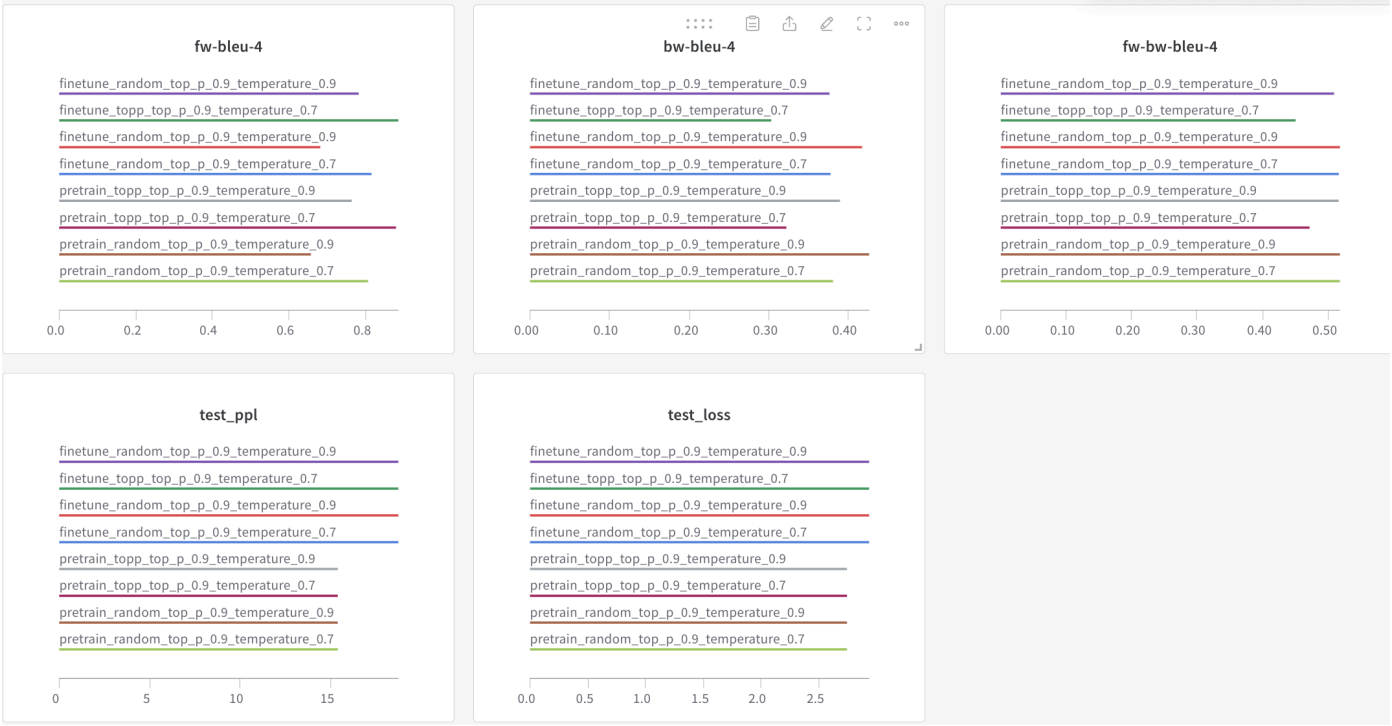
基本模型

训练结果如下



metric	Tfmr_scratch	Tfmr_finetune
train-loss	**2.07149**	2.17592
var-loss	3.16324	**2.95237**
var-ppl	23.64712	**19.151**
bw-bleu-4	0.42531	**0.43419**
fw-bleu-4	**0.58346**	0.56896
har-bleu-4	0.49199	**0.49252**
test-loss	2.93128	**2.73662**
ppl	18.7516	**15.43472**

推理策略及温度实验



model	strategy	temperature	forward BLEU-4	backward BLEU-4	harmonic BLEU-4
Tfmr_scratch	random	1	0.806	0.381	**0.518**
Tfmr_scratch	random	0.7	0.659	**0.426**	0.517
Tfmr_scratch	topp	1	0.879	0.322	0.472
Tfmr_scratch	topp	0.7	0.764	0.389	0.515
Tfmr_finetune	random	1	0.817	0.378	0.517
Tfmr_finetune	random	0.7	0.683	0.417	**0.518**
Tfmr_finetune	topp	1	**0.886**	0.303	0.451
Tfmr_finetune	topp	0.7	0.784	0.376	0.508

语句分析

我 `rand` 了十条语句，下标如下

```
[920, 2406, 1961, 3667, 107, 7, 4903, 655, 4870, 2456]
```

我将按照 Ground truth, Tfmr_scratch, Tfmr_finetune 的顺序给出语句

```
Ground_truth:
==================================================
A silver and clear drink container with a black tray of fruit next to it .

--------------------------------------------------
A dirt bike is parked next to a wooden wall .

--------------------------------------------------
```

```
A small slice of cake sits on a plate with a fork .

--------------------------------------------------
An airplane is on the ground near a grassy field and forest of trees .

--------------------------------------------------
Two people putting food to be cooked into two different ovens .

--------------------------------------------------
A crowd of skiers prepare to ski down snow covered slopes .

--------------------------------------------------
Empty park benches positioned at the edge of a lake .

--------------------------------------------------
Two bikes leaning against the railing of an area looking over some water .

--------------------------------------------------
Two giraffes are standing next to each other by a tree .

--------------------------------------------------
A man riding a motorcycle while wearing a leather jacket .

==================================================

Tfmr_scratch:
==================================================
A bench sits and next to a tree in a park .
--------------------------------------------------
A couple of men sitting on top of a bench .
--------------------------------------------------
A man and a woman standing next to a young boy sitting on a bench .
--------------------------------------------------
A giraffe standing in a field full of wild animals .
--------------------------------------------------
A sidewalk and street with buildings , a bus drives down it .
--------------------------------------------------
A bus stopping at an American story stopped at a railroad crossing street .
--------------------------------------------------
Two people are on a street corner next to a fire hydrant .
--------------------------------------------------
A bus on a rain soaked street front of a church .
--------------------------------------------------
A giraffe and two small zebra in a grassy field .
--------------------------------------------------
A male and yellow bus driving a rider that shopicolored styles .
==================================================

Tfmr_finetune:
==================================================
A statue of several electronic boxes sitting in a office sitting near a bench .
--------------------------------------------------
```

```
A couple rides past a traffic light opposite directions .
--------------------------------------------------
A man and a woman lying beside a wooden bench .
--------------------------------------------------
A large plane sits in a field full of trees .
--------------------------------------------------
A sidewalk along a street with a horse stopping light .
--------------------------------------------------
A bus driving past a rural area covered in snow .
--------------------------------------------------
Two people standing on front of a kite and a fence as a planter with trees in the
background .
--------------------------------------------------
A bus on a rain soaked street front of a bus stop .
--------------------------------------------------
A giraffe and a small field with several other buildings .
--------------------------------------------------
A yellow bus is travelling down a street that is stopped with cars and river .
==================================================
```

1. 语法错误：

Tfmr_scratch：

- “A bench sits and next to a tree in a park.”（“sits and”使用不当；应为“A bench sits next to a tree in a park.”）
- “A couple of men sitting on top of a bench.”（主谓不一致；应为“Two men are sitting on top of a bench.”）
- “A man and a woman standing next to a young boy sitting on a bench.”（主谓不一致；应为“A man and a woman are standing next to a young boy sitting on a bench.”）
- “A giraffe standing in a field full of wild animals.”（主谓不一致；应为“A giraffe is standing in a field full of wild animals.”）
- “A bus stopping at an American story stopped at a railroad crossing street.”（语法错误和意义不明确；应为“A bus stops at a railroad crossing on a street.”）
- “A male and yellow bus driving a rider that shopicolored styles.”（语法错误和意义不明确；应为“A yellow bus is driving, carrying a rider in colorful clothing.”）

Tfmr_finetune：

- “A statue of several electronic boxes sitting in a office sitting near a bench.”（语法错误；应为“A statue of several electronic boxes sits in an office near a bench.”）
- “A couple rides past a traffic light opposite directions.”（意义不明确；应为“A couple rides past a traffic light in opposite directions.”）
- “A large plane sits in a field full of trees.”（主谓不一致；应为“A large plane sits in a field full of trees.”）
- “A sidewalk along a street with a horse stopping light.”（意义不明确；应为“A sidewalk along a street with a horse crossing light.”）
- “A bus driving past a rural area covered in snow.”（主谓不一致；应为“A bus drives past a rural area covered in snow.”）

- “A yellow bus is travelling down a street that is stopped with cars and river.”（意义不明确；应为“A yellow bus is traveling down a street that is stopped due to cars and a river.”）

2. 最佳句子策略/模型：
   Tfmr_finetune 的句子虽然仍然包含一些错误，但总体上比Tfmr_scratch的句子更连贯，更接近Ground_truth，所以更好。
3. 与指标的一致性：

如果只看句子质量，我会认为 Tfmr_finetune 指标都更高：

- 困惑度可能对 Tfmr_finetune 最低，因为它的清晰度和准确性。
- 前向 BLEU 和后向 BLEU 也对 Tfmr_finetune 更高，表明与参考句子的匹配度更高。
- 调和 BLEU 也对 Tfmr_finetune 最高，反映了在翻译质量方面的整体表现更好。

但实际上 Tfmr_scratch 的 fw-bleu 更高，在这里是不一致的，其余一致。不过考虑数据量很小，baseline 稍微有一点更胜一筹也是合理的。

正则化方式比较

1. Post-Norm version of LayerNorm.[7]

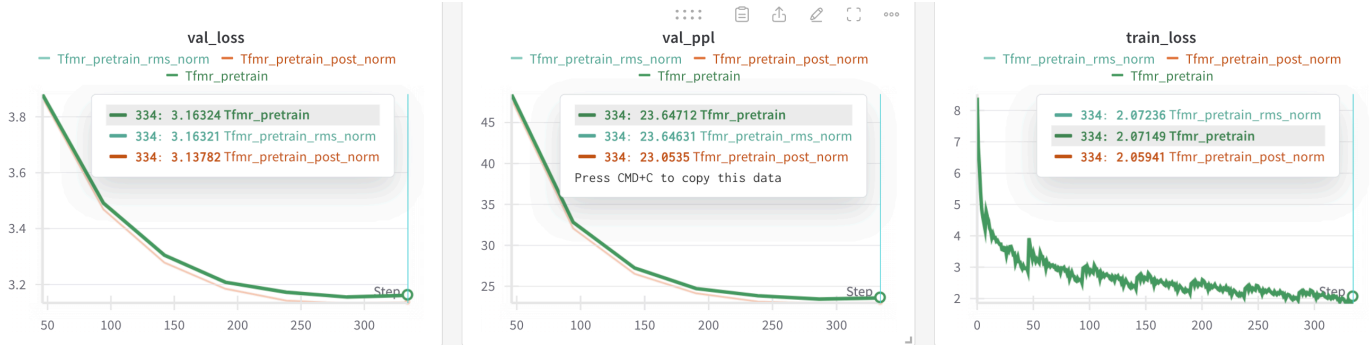
$$
\begin{aligned}
Pre\ Norm: x_{t+1} &= x_t + F_t(Norm(x_t)) \\
Post\ Norm: x_{t+1} &= Norm(x_t + F_t(x_t))
\end{aligned} \tag{1}
$$

2. RMSNorm [8]:

$$
\bar{a}_i = \frac{a_i}{\mathrm{RMS}(a)} g_i, \quad \text{where} \quad \mathrm{RMS}(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}. \tag{2}
$$

a_i is the i-th value of vector a, and g_i is the gain parameter used to re-scale the standardized value.

首先，结果如下，看上去三者非常接近，但是 `PreNorm` 的 `loss` 和 `ppl` 稳定略大一些，然后 `RMSNorm` 相对二者大概快了 17%。



这是我对 `RMSNorm` 和 `PostNorm` 的实现

```
class RMSNorm(nn.Module):
    def __init__(self, hidden_size, eps=1e-6):
        super().__init__()
        self.weight = nn.Parameter(torch.ones(hidden_size))
        self.eps = eps

    def forward(self, hidden_states):
        variance = hidden_states.to(torch.float32).pow(2).mean(-1, keepdim=True)
        hidden_states = hidden_states * torch.rsqrt(variance + self.eps)
        return self.weight * hidden_states
self.ln_1 = RMSNorm(hidden_size, eps=config.layer_norm_epsilon) if NORM == "rms" else nn.LayerNorm(hidden_size, eps=config.layer_norm_epsilon)


if NORM == "pre":
        residual = hidden_states
        hidden_states = self.ln_2(hidden_states)
        feed_forward_hidden_states = self.mlp(hidden_states)
        # residual connection
        hidden_states = residual + feed_forward_hidden_states
elif NORM == "post":
        residual = hidden_states
        # residual connection
        hidden_states = self.ln_2(residual + self.mlp(hidden_states))
elif NORM == "rms":
        residual = hidden_states
        hidden_states = self.ln_2(hidden_states)
        feed_forward_hidden_states = self.mlp(hidden_states)
        hidden_states = residual + feed_forward_hidden_states
```

分析如下，

首先对于 PreNorm 和 PostNorm：

- 训练稳定性：PreNorm 由于在每一层都进行归一化，因此对于梯度消失和爆炸的问题缓解较好，使得训练过程更加稳定
- PostNorm 因为归一化位置改变，需要更仔细的参数初始化和学习率调整，大规模上必须要调 warm-up
- 方差累计：PreNorm 中，由于每一层的输入都经过归一化，方差会累积在残差连接中，可能导致深层网络的训练效果类似于扩展了模型宽度，而不是真正的深度

- 最终效果：尽管 PreNorm 在训练过程中更稳定，但 PostNorm 在微调后的效果通常更好，与实验预期一致

再看额外的 RMSNorm：

- 计算效率：RMSNorm 是 LayerNorm 的一种变体，它通过计算均方根来归一化输入，而不是使用方差。这种简化使得 RMSNorm 在计算速度上比 LayerNorm 快

性能差异：

- 收敛性能：PreNorm 更有利于模型收敛，缓解梯度消失/爆炸的问题。PostNorm 的收敛性略差，梯度问题更严重
- 训练稳定性：PreNorm 对参数初始化和学习率更鲁棒，而 PostNorm 更依赖参数初始化，需要小心调参
- 计算效率：PreNorm 每层都需计算规范化，计算量大。PostNorm 只在层间计算一次规范化，计算量小；RMSNorm 更小

最终结果

我使用了 Tfmr_finetune，其中温度设为 0.7，使用 random 解码策略，其余参数不变。

model	strategy	temperature	forward BLEU-4	backward BLEU-4	harmonic BLEU-4
Tfmr_finetune	random	0.7	0.683	0.417	**0.518**

三个问题

Transformer 与 RNN 的比较

- 时间复杂度：RNN 由于其序列处理的特性，需要逐个处理序列元素，因此时间复杂度是 $O(T \cdot N)$，其中 T 是序列长度，N 是隐藏层的维度。而 Transformer 则可以并行处理序列中的所有元素，其时间复杂度为 $O(N \cdot d)$，其中 d 是自注意力层的计算复杂度。
- 空间复杂度：RNN 的空间复杂度主要取决于隐藏层的维度，而 Transformer 除了需要存储隐藏层状态外，还需要存储自注意力层的计算结果，因此空间复杂度相对较高。
  -性能：Transformer 在处理长距离依赖的任务上通常比RNN表现更好，因为它可以更好地捕捉序列中的长距离关系。
- 位置编码：RNN通常不需要位置编码，因为它的递归结构自然地保留了序列的顺序信息。而 Transformer 由于缺乏递归结构，需要显式的位置编码来提供序列中单词的位置信息（目前比较新的是 RoPE，苏神?）。

推理时间复杂度：

1. 在 model_tfmr.py 中设置 `use_cache=True` 的目的是为了缓存自注意力层的计算结果，以便在解码时可以复用，减少重复计算。如果设置为 False，则每次解码时都会重新计算自注意力层的结果，这会增加推理时间。
2. 我们这里只考虑一个块（即 $B = 1$，否则乘上 B 即可）的情况。解码第 t 个 token 的时间复杂度主要由自注意力层和前馈层决定。自注意力层的时间复杂度为 $O(T \cdot d \cdot n)$，前馈层的时间复杂度为 $O(d^2)$。因此，解码第 t 个token的时间复杂度为 $O(T \cdot d \cdot n + d^2)$。对于整个序列 L，总的时间复杂度为 $O(T \cdot (T \cdot d \cdot n + d^2))$。自注意力模块在序列长度 T 较大时占主导地位，因为其时间复杂度与序列长度成线性关系。而前馈层在隐藏层维度 d 较大时占主导地位，因为其时间复杂度与 d 的平方成正比。

3. 预训练可以显著提高模型的生成质量，因为它提供了丰富的先验知识，帮助模型更好地理解语言结构和语义信息。同时，预训练还可以加快模型的收敛速度，因为预训练模型已经学习到了数据中的一些通用特征。根据实验设置（训练任务、数据、预训练检查点等），预训练的影响是否符合预期取决于预训练数据的质量和相关性。如果预训练数据与目标任务数据相似，那么预训练的影响通常会非常显著。反之，如果预训练数据与目标任务数据差异较大，预训练的影响可能会有限。

bpe 解码和split 的区别

实现 GQA

从12层数据里面拿不同层

Bonus1 BPE 分词实验

测试代码如下：

```
if __name__ == "__main__":
    import os
    models_dir = 'tokenizer'
    tokenizer = get_tokenizer(models_dir)
    text = "This is a test sentence for BPE tokenizer."
    tokens = tokenizer.tokenize(text)
    print("Raw Text:", text)
    print("BPE Tokenized Text:", tokens)
    encoded_tokens = tokenizer.encode(text)
    print("BPE Encoded Tokens:", encoded_tokens)
    decoded_text = tokenizer.decode(encoded_tokens)
    print("Decoded Text:", decoded_text)
```

测试结果如下

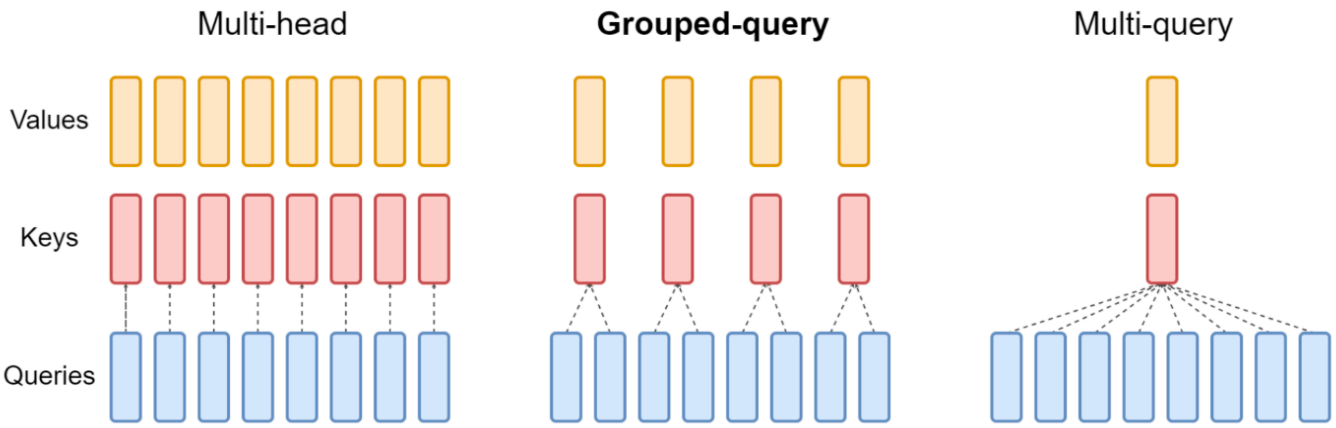
```
Raw Text: This is a test sentence for BPE tokenizer.
BPE Tokenized Text: ['This', '?is', '?a', '?test', '?sentence', '?for', '?B', 'PE', '?token', 'izer', '.']
BPE Encoded Tokens: [1212, 318, 257, 1332, 6827, 329, 347, 11401, 11241, 7509, 13]
Decoded Text: This is a test sentence for BPE tokenizer.
```

BPE 的优势：

1. 处理未知词汇：基于空格的分词通常假设文本已经被预先分割成单词或词汇，这在处理一些没有明显单词边界的语言（如中文）时可能不太适用。即使在有明显单词边界的语言中，对于未知词汇或新出现的词汇，基于空格的分词可能无法正确处理。BPE 分词器通过学习最常见的字符对，并将其合并为单个单元，可以更有效地处理未知词汇。
2. 提高词汇表示的一致性：在基于空格的分词中，相同的字符序列在不同的上下文中可能会被分割成不同的词汇，这会导致相同的概念在模型中有不同的表示。BPE分词器通过学习字符对的合并规则，可以在不同的上下文中保持词汇表示的一致性，从而提高模型的性能。
3. 减少词汇表的大小：BPE分词器通过合并常见的字符对来创建子词单元，这可以显著减少模型需要学习的词汇表大小。相比之下，基于空格的分词可能需要一个非常大的词汇表来覆盖所有可能的词汇。较小的词汇表可以减少模型的内存占用，加快训练和推理速度，同时也可以减轻存储和传输数据的负担。

4. 提高模型的泛化能力：BPE分词器生成的子词单元可以捕捉到词汇的内部结构，这有助于模型更好地理解和泛化词汇的变化，例如词形变化、拼写错误和新词。相比之下，基于空格的分词可能无法捕捉到这些细微的词汇变化，从而影响模型的泛化能力。

Bonus2 GQA 实现



参考 TfmrAttention，我的实现如下

```
class GQAttention(nn.Module):
    def __init__(self, config):
        super().__init__()

        max_positions = config.max_position_embeddings
        self.register_buffer(
            "bias",
            torch.tril(torch.ones((max_positions, max_positions), dtype=torch.uint8).view(
                1, 1, max_positions, max_positions
            ))
        )
        self.register_buffer("masked_bias", torch.tensor(-1e4))

        config.num_key_value_groups = config.num_attention_heads // 4
        self.embed_dim = config.hidden_size
        self.num_heads = config.num_attention_heads
        self.head_dim = self.embed_dim // self.num_heads
        self.num_groups = config.num_key_value_groups
        self.num_kv_heads = self.num_heads // self.num_groups

        if self.head_dim * self.num_heads != self.embed_dim:
            raise ValueError(
                f"`embed_dim` must be divisible by num_heads (got `embed_dim`: {self.embed_dim} and `num_heads`: {self.num_heads})."
            )

        self.scale_attn_weights = config.scale_attn_weights
        self.q_proj = TransposeLinear(self.embed_dim, self.embed_dim)
```

```
        self.k_proj = TransposeLinear(self.embed_dim // self.num_groups, self.embed_dim)
        self.v_proj = TransposeLinear(self.embed_dim // self.num_groups, self.embed_dim)
        self.o_proj = TransposeLinear(self.embed_dim, self.embed_dim)

        self.attn_dropout = nn.Dropout(config.attn_pdrop)
        self.resid_dropout = nn.Dropout(config.resid_pdrop)

    def _attn(self, query, key, value):
        attn_weights = query @ key.transpose(-1, -2)

        if self.scale_attn_weights:
            attn_weights = attn_weights / (float(value.size(-1)) ** 0.5)

        causal_mask = self.bias[..., key.size(-2) - query.size(-2): key.size(-2), :key.size(-2)]
        attn_weights = attn_weights * causal_mask + self.masked_bias.to(attn_weights.device).to(attn_weights.dtype) * (1 - causal_mask)

        attn_weights = nn.functional.softmax(attn_weights, dim=-1)
        attn_weights = self.attn_dropout(attn_weights)
        attn_output = attn_weights @ value

        return attn_output, attn_weights

    def _split_heads(self, tensor, num_heads, attn_head_size):
        assert tensor.size(-1) == num_heads * attn_head_size, f"hidden_size: {tensor.size(-1)} != num_heads * attn_head_size: {num_heads * attn_head_size}"
        x_shape = tensor.size()[: -1] + (num_heads, attn_head_size)
        return tensor.view(*x_shape).permute(0, 2, 1, 3)

    def _merge_heads(self, tensor, num_heads, attn_head_size):
        tensor = tensor.permute(0, 2, 1, 3).contiguous()
        x_shape = tensor.size()[:-2] + (num_heads * attn_head_size,)
        return tensor.view(*x_shape)


    def forward(
        self,
        hidden_states,
        layer_past=None,
        use_cache=False,
    ):
        query = self.q_proj(hidden_states)
        key = self.k_proj(hidden_states)
        value = self.v_proj(hidden_states)

        query = self._split_heads(query, self.num_heads, self.head_dim)
        key = self._split_heads(key, self.num_kv_heads, self.head_dim)
        value = self._split_heads(value, self.num_kv_heads, self.head_dim)

```

```
        # Repeat k and v for each group
        key = key.repeat_interleave(self.num_groups, dim=1)
        value = value.repeat_interleave(self.num_groups, dim=1)

        if layer_past is not None:
            past_key, past_value = layer_past
            key = torch.cat((past_key, key), dim=-2)
            value = torch.cat((past_value, value), dim=-2)

        if use_cache is True:
            present = (key, value)
        else:
            present = None

        attn_output, attn_weights = self._attn(query, key, value)

        attn_output = self._merge_heads(attn_output, self.num_heads, self.head_dim)
        attn_output = self.o_proj(attn_output)
        attn_output = self.resid_dropout(attn_output)

        outputs = (attn_output, present)
        outputs += (attn_weights,)

        return outputs  # a, present, (attentions)
```

Metric	Tfmr	GQA-2grp	GQA-4grp
bw-bleu-4	0.42531	0.42715	**0.42992**
fw-bleu-4	0.58346	0.57913	**0.58725**
har-bleu-4	**0.49199**	0.49013	0.49042
test-loss	2.93128	**2.83631**	2.91684
ppl	18.7516	**18.69218**	18.48279

我发现，每次聚合两个 Attention Header，大概能减少 120M 内存占用，推导如下（参考 Kimi，修了一些不对的地方）。

首先，在 Transformer 模型中，Multi-Head Attention (MHA) 的内存占用主要来自于存储所有的 query、key、和 value 向量。如果我们使用 Grouped-Query Attention (GQA)，我们可以减少存储的 key 和 value 向量的数量，因为它们会被多个 query 头共享。

假设我们有以下参数：

- H：原始的头数（对于MHA）
- D：每个头的维度
- N：序列长度
- G：GQA中的组数

在 MHA 中，每个头都有自己的一套 key 和 value 向量，所以总的内存占用为：$3 \times H \times D \times N$

而在GQA中，我们把头分成了 G 组，每组共享一个 key 和一个 value 向量。因此，每个组只需要存储一个 key 和一个 value 向量，而每个头仍然有自己的 query 向量。所以 GQA 的内存占用为：
$(G \times 2 \times D + H \times D) \times N = (2 \times D \times G + D \times H) \times N$

如果我们假设每个组包含 $\frac{H}{G}$ 个头，那么内存占用可以简化为：

$$\left(2 \times D \times \frac{H}{G} + D \times H\right) \times N = \left(\frac{2DH}{G} + DH\right) \times N = DH \times \left(\frac{2}{G} + 1\right) \times N$$

内存减少量可以计算为 MHA 和 GQA 内存占用的差：

$$3 \times H \times D \times N - DH \times \left(\frac{2}{G} + 1\right) \times N = DH \times N \times \left(3 - \left(\frac{2}{G} + 1\right)\right) = DH \times N \times \left(2 - \frac{2}{G}\right)$$

内存减少的比例则是：

$$\frac{DH \times N \times \left(2 - \frac{2}{G}\right)}{3 \times H \times D \times N} = \frac{2 - \frac{2}{G}}{3}$$

从上面的公式可以看出，随着组数 G 的增加，内存减少的比例会降低。当 G 接近 H 时，内存减少的比例接近 $\frac{2}{3}$，这意味着 GQA 的内存占用接近 MHA。而当 G 为1时，即 MQA 的情况，内存减少的比例最大，理论上可以减少到原来的 $\frac{1}{3}$。