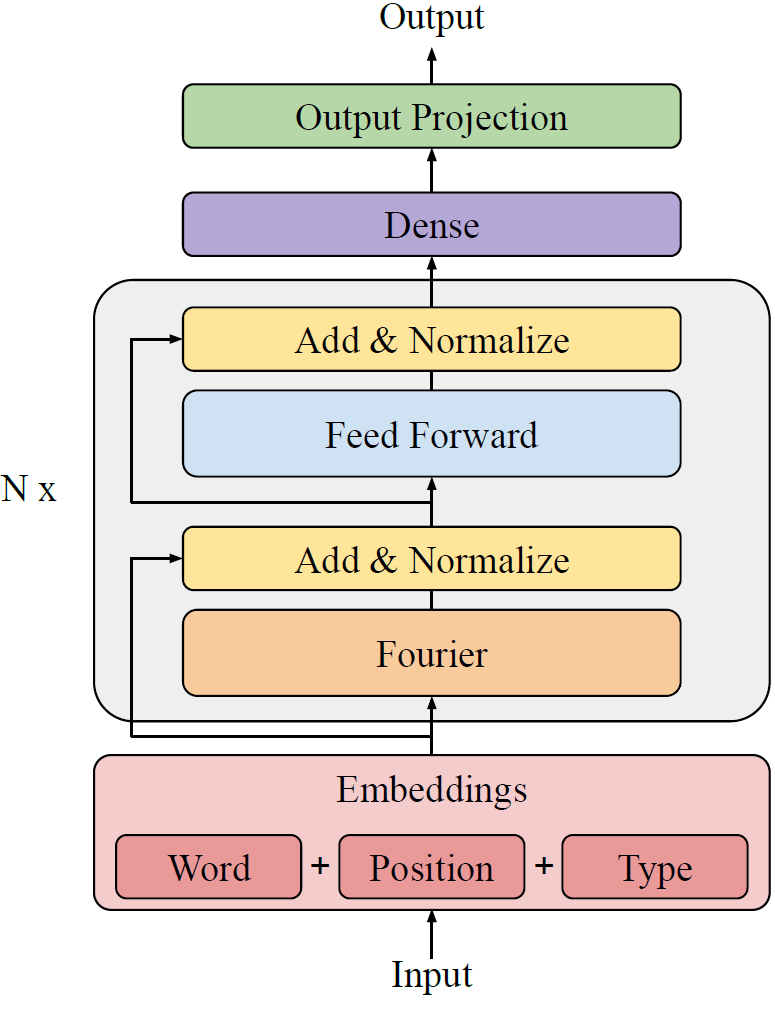
复现相关总结：

在掌握Fnet的基础上，开始在原论文的基础上复现并进一步改进和应用，首先介绍一下将Fnet替换attention后加入到transformer中后的示例架构，如下图\*所示：



Fnet模型关键代码：

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | **#前馈神经网络层：在每个 FNet 块中对输入进行进一步的特征变换和非线性处理**  **class** FeedForward(nn.Module):  **def** \_\_init\_\_(self, dim, hidden\_dim, dropout **=** 0.):           super().\_\_init\_\_()           self.net **=** nn.Sequential(               nn.Linear(dim, hidden\_dim), **# 全连接层，将输入维度映射到隐藏层维度**               nn.GELU(), **# GELU 激活函数，增加非线性**               nn.Dropout(dropout), **# Dropout 层，防止过拟合**               nn.Linear(hidden\_dim, dim), **# 再次全连接层，将隐藏层维度映射回输入维度**               nn.Dropout(dropout) **# 另一个 Dropout 层**           )  **def** forward(self, x):  **return** self.net(x)  **#归一化：对输入进行层归一化，稳定和加速训练过程**  **class** PreNorm(nn.Module):  **def** \_\_init\_\_(self, dim, fn):           super().\_\_init\_\_()           self.norm **=** nn.LayerNorm(dim)           self.fn **=** fn  **def** forward(self, x, **\*\***kwargs):  **return** self.fn(self.norm(x), **\*\***kwargs) **# 先进行层归一化，再应用给定的函数**  **#FNet块：实现了 FNet 的核心操作，使用傅里叶变换来混合输入序列中的标记**  **class** FNetBlock(nn.Module):  **def** \_\_init\_\_(self):        super().\_\_init\_\_()  **def** forward(self, x):  **# 双维度傅里叶变换，返回实部**       x **=** torch.fft.fft(torch.fft.fft(x, dim**=-**1), dim**=-**2).real  **return** x  **#完整的架构如下：**  **class** FNet(nn.Module):  **def** \_\_init\_\_(self, dim, depth, mlp\_dim, dropout **=** 0.):           super().\_\_init\_\_()           self.layers **=** nn.ModuleList([])  **for** \_ **in** range(depth):               self.layers.append(nn.ModuleList([  **# 包装 FNetBlock 的 PreNorm**                   PreNorm(dim, FNetBlock()),  **# 包装 FeedForward 的 PreNorm**                   PreNorm(dim, FeedForward(dim, mlp\_dim, dropout **=** dropout))               ]))  **def** forward(self, x):  **for** attn, ff **in** self.layers:               x **=** attn(x) **+** x # 残差连接：加上输入               x **=** ff(x) **+** x  **return** x | |

FNet之所以能够替代attention的原因，是因为以下几点：

* **全局信息捕捉**：FNet 使用傅里叶变换来混合输入序列中的标记，从而捕捉序列中的全局信息。相比之下，虽然可以通过自注意力机制捕捉长距离依赖关系，但其计算复杂度较高，尤其是在处理长序列时。傅里叶变换具有线性复杂度，可以更有效地处理全局信息。
* **计算效率：**注意力机制的计算复杂度随着序列长度的增加而增加，而傅里叶变换的计算复杂度是线性的，因此在处理长序列时，FNet 的计算效率更高。
* **参数效率：**FNet 不需要像注意力机制那样依靠大量的参数（QKV矩阵）来学习权重。而是通过固定的傅里叶变换来捕捉序列中的全局信息，因此具有更好的参数效率。