RWKV 介绍

RWKV 是一种新型的 transformer 模型，最早被当作一种语言模型提出，现在在计算机视觉 、多模态等领域也表现出了相当不错的应用潜力。相比于以往的传统 Transformer 模型和 RNN 模型，RWKV 主要具备以下几个优势：

- **注意力计算的复杂度低。**传统 Transformer 在计算自注意力的时候，假设序列的长度是 N ,那么它的计算复杂度就是 $O(N^2)$ ，这样就导致 Transformer 很难处理长序列的问题，一般只能将长序列切分成若干较短的序列（segment）分开处理。而 RWKV 提出了 Time Mix 模块来代替传统的自注意力，将计算复杂度降低为 $O(N)$ ，大大提升了 RWKV 处理长序列的能力。
- **运行速度较快。**RNN 在计算一个序列的 hidden state 时，下一个输入 token 的 hidden state 需要基于上一个输入 token 的 hidden state 进行计算。这就使得 RNN 内部的计算只能是串行的，无法高效地并行计算。而 RWKV 利用了自注意力的机制，使得计算所有 token 的 hidden state 可以并行地进行，大大提高了计算的速度。
- **拟合性能好。**RNN 最大的问题在于最终的拟合性能不如 Transformer，然而 RWKV 则可以在继承了 RNN 优势的同时，比肩甚至超越传统 Transformer 的拟合性能。

总而言之，RWKV 弥补了上述传统 Transformer 和 RNN 各自的劣势，表现出比较亮眼的效果。通过下图可以看出，在参数量和 GPT 和 Pythia 相似的时候，RWKV 在下游任务上，可以比肩甚至超越这些模型的性能。同时，RWKV 计算每个 token 的时间相比于 GPT 大约下降了 54%，显存消耗降低了大约 19%，大大节约了计算资源的开销。

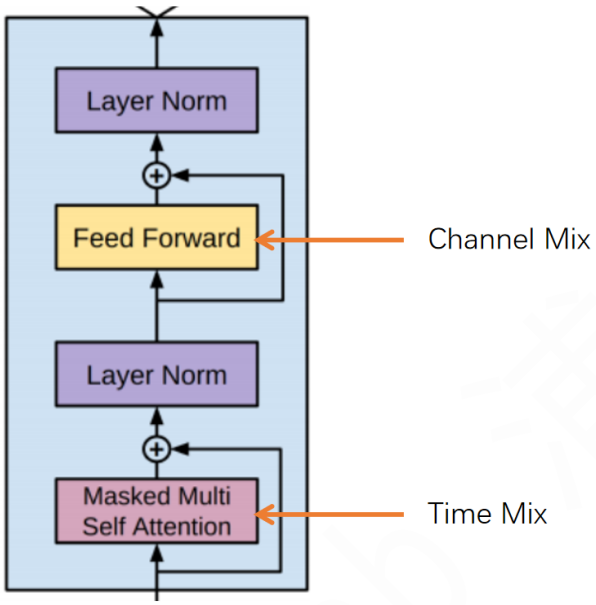
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
	B	ppl	%	acc	acc	acc	acc_norm	acc	acc_norm	acc	acc_norm	acc_norm	acc	acc	em	acc
RWKV-4	0.17	29.33	44.13%	32.99%	65.07%	58.79%	32.26%	50.83%	24.15%	47.47%	25.78%	29.60%	77.50%	1.26%	62.03%	66.00%
Pythia	0.16	24.38	44.14%	38.97%	62.68%	58.47%	31.63%	52.01%	23.81%	45.12%	25.82%	29.20%	76.50%	1.31%	66.32%	62.00%
GPT-Neo	0.16	30.27	43.42%	37.36%	63.06%	58.26%	30.42%	50.43%	23.12%	43.73%	25.16%	26.20%	76.60%	1.18%	64.92%	64.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4	0.43	13.04	48.04%	45.16%	67.52%	63.87%	40.90%	51.14%	25.17%	52.86%	27.32%	32.40%	80.30%	2.35%	70.48%	65.00%
Pythia	0.4	11.58	48.39%	50.44%	66.70%	62.64%	39.10%	53.35%	25.77%	50.38%	25.09%	30.00%	81.50%	2.03%	75.05%	67.00%
GPT-Neo	0.4	13.88	47.25%	47.29%	65.07%	61.04%	37.64%	51.14%	25.34%	48.91%	26.00%	30.60%	81.10%	1.38%	73.79%	65.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4	1.5	7.04	53.91%	56.43%	72.36%	68.73%	52.48%	54.62%	29.44%	60.48%	27.64%	34.00%	85.00%	5.65%	76.97%	77.00%
Pythia	1.4	6.58	53.55%	60.43%	71.11%	67.66%	50.82%	56.51%	28.58%	57.74%	27.02%	30.80%	85.50%	5.52%	81.43%	73.00%
GPT-Neo	1.4	7.50	52.64%	57.25%	71.16%	67.72%	48.94%	54.93%	25.85%	56.19%	27.86%	33.60%	86.00%	5.24%	80.62%	69.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4	3	5.24	57.52%	63.94%	73.72%	70.28%	59.63%	59.43%	31.83%	64.27%	28.74%	37.60%	85.70%	11.07%	80.56%	81.00%
RWKV-4	3, ctx4k	5.25	57.93%	63.96%	74.16%	70.71%	59.89%	59.59%	33.11%	65.19%	28.45%	37.00%	86.50%	11.68%	80.87%	82.00%
Pythia	2.8	4.93	57.64%	65.36%	73.83%	70.71%	59.46%	61.25%	32.25%	62.84%	28.96%	35.20%	87.70%	9.63%	85.10%	77.00%
GPT-Neo	2.8	5.63	55.92%	62.22%	72.14%	69.54%	55.82%	57.62%	30.20%	61.07%	27.17%	33.20%	89.30%	4.82%	83.80%	80.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
RWKV-4	7.4	4.38	61.20%	67.18%	76.06%	73.44%	65.51%	61.01%	37.46%	67.80%	31.22%	40.20%	88.80%	18.30%	83.68%	85.00%
Pythia	6.9	4.30	60.44%	67.98%	74.54%	72.96%	63.92%	61.01%	35.07%	66.79%	28.59%	38.00%	90.00%	15.42%	86.44%	85.00%
GPT-J	6.1	4.10	61.34%	68.31%	75.41%	74.02%	66.25%	64.09%	36.60%	66.92%	28.67%	38.20%	91.50%	16.74%	87.71%	83.00%

接下来，我们将详细介绍 RWKV 的技术部分。

概要

RWKV 总体遵循 Transformer 的模型架构。如下图所示，左侧的方框内是标准 Transformer 的一个块，大致包括：自注意力层、Layer Norm层、前馈神经网络、Layer Norm层四个部分。而 RWKV 就是针对这个结构做了改进，使用 Time Mix 模块代替了自注意力层，使用 Channel Mix 模块代替了前馈神经网络。一句话概括就是：

$$\mathrm{RWKV} = \mathrm{TimeMix} + \mathrm{ChannelMix}$$



这里再介绍一下为什么 RWKV 提出的这两个模块要起这样的名字。这是因为：

- 自注意力层的本质，就是让不同 token 的信息互相交流，这其实就是在时间维度交流信息。而作者在设计 RWKV 的新模块时，没有改变这种作用的本质，因此给新的模块起名为 Time Mix。
- 而前馈神经网络的本质，就是若干个线性层，让同一个 token 的不同通道之间进行信息交流。同理，作者给替换这一功能的模块起名为 Channel Mix。

接下来，我们就将对这两个改进模块的具体设计做进一步说明。

Time Mix

在这个模块中，作者主要提出了两个针对原始自注意力的改进：Time Shift、RWKV Attention，我们逐一进行分析。

Time Shift

对于每一层 transformer 块的输入 token 序列，RWKV 的设计是让它们首先通过一个叫做 Time Shift 的操作。这个操作的具体实现如下：

假设下方的矩阵代表了一个序列的输入，其中矩阵的每一行代表一个 token，每一列则代表一个 channel。

$$\begin{matrix} x_{01} & x_{02} & x_{03} & x_{04} \\ x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{matrix}$$

而 Time Shift 就是对此输入进行了一次“移位”变换，结果如下：

$$\begin{matrix} x_{01} & x_{02} & 0 & 0 \\ x_{11} & x_{12} & x_{03} & x_{04} \\ x_{21} & x_{22} & x_{13} & x_{14} \\ 0 & 0 & x_{23} & x_{24} \end{matrix}$$

简单来说，通过这个移位操作，token[t] 一半的通道，被 token[t-1] 一半的通道代替了。

作者通过实验表明，这个 Time Shift 操作不仅使用于 RWKV 结构，任意的 Transformer 结构在使用了这一操作之后，都可以得到性能的提升。那么为什么这样做会产生好处呢，作者给出了三个解释：

- **添加了 inductive bias。**Time Shift 的方式更像是添加了一种人为的先验，强制让每个 token 生成的 hidden state 包含历史的语境信息。
- **明确了信息分离。**Hidden state 的通道主要做了两个事情：根据本 token 的信息预测下一个字，收集前文的语境信息，传递给后文。而这两个内容，刚好对应了没有 Time Shift 的通道和有 Time Shift 的通道。这样一来就做到了信息的分离。
- **类似 RNN 的信息传递。**在 RNN 中，一个 token 计算出来的 hidden state 信息会被用于计算本层下一个 token 的 hidden state。而在加入了 Time Shift 之后，一个 token 计算出来的 hidden state 信息，则会被用于计算下一层下一个 token 的 hidden state。这种类似 RNN 信息传递的方式，也可能是帮助 Transformer 进行学习的潜在原因之一。

RWKV Attention

这一部分，RWKV 主要是针对标准的 attention，进行了修改，大大减少了计算量的同时，没有降低原有的性能。

首先，我们先回顾一下标准的 attention 计算公式，对于第 t 个 token 生成的 hidden state $H[t]$ ，有如下的计算公式：

$$\mathbf{H}[t] = \frac{\sum_{i=1}^{t} e^{\mathbf{Q}_t \cdot \mathbf{K}_i / \sqrt{d}} \cdot \mathbf{V}_i}{\sum_{i=1}^{t} e^{\mathbf{Q}_t \cdot \mathbf{K}_i / \sqrt{d}}}$$

其中 $\mathbf{Q}_i$ $\mathbf{K}_i$ $\mathbf{V}_i$ 分别指由输入的第 i 个 token 经过线性层变化得来的张量，$\cdot$ 则指的是矩阵的点乘运算。

而 RWKV attention 则是在此基础上进行了改进：

$$\mathbf{H}[t] = \sigma(\mathbf{R}_{t-1}) \frac{\sum_{i=1}^{t} e^{(t+1-i)\mathbf{W} + \mathbf{K}_i} \cdot \mathbf{V}_i}{\sum_{i=1}^{t} e^{(t+1-i)\mathbf{W} + \mathbf{K}_i}}$$

其中 $\mathbf{R}_i$ $\mathbf{K}_i$ $\mathbf{V}_i$ 分别指由输入的第 i 个 token 经过线性层变化得来的张量，$\mathbf{W}$ 则指的是一个可学习的参数。通过观察公式可以看出，由于去掉了复杂的矩阵点乘计算，RWKV attention 确实大大简化了计算复杂度，使得 attention 的计算量降至随序列长度线性增长。

这里有一个比较有意思的地方在于，作者对 $\mathbf{W}$ 前乘了一个标量 t+1-i 。作者将这个技巧称作 Time Weighting，其设计的动机在于，要使得距离当前 token 不同距离的历史 token 对输出的贡献不同。这个技巧最早并非在 RWKV 中提出，详细的设计细节可以查看参考链接。

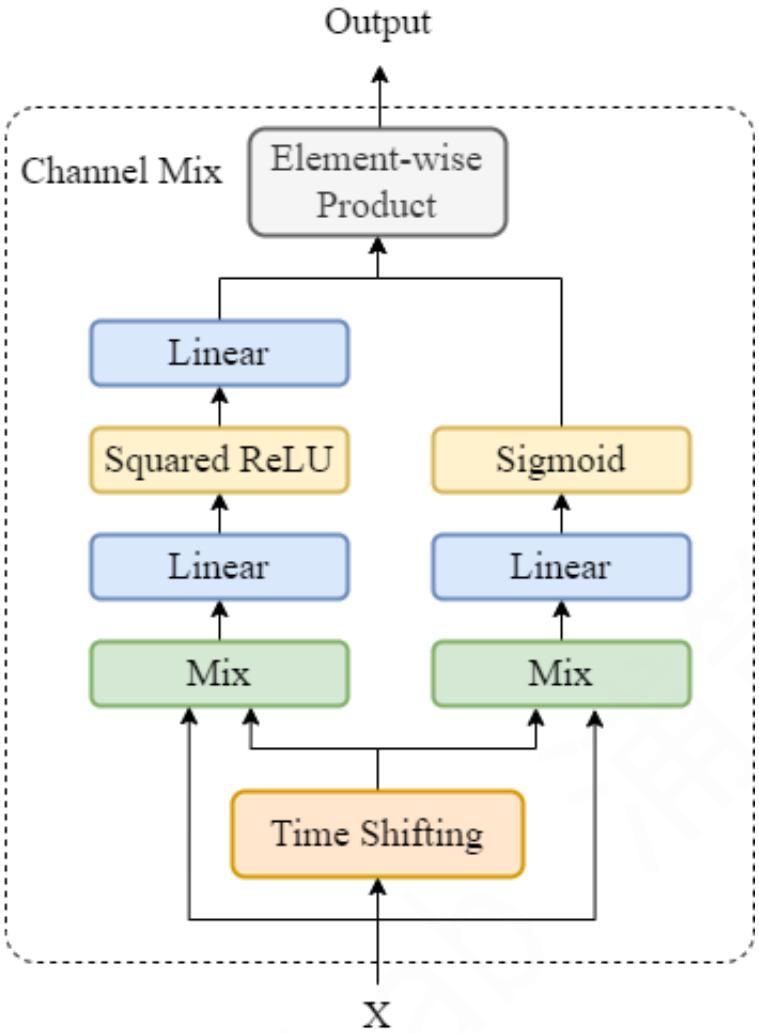
接下来是该模块的伪代码：

```
def forward(self, x):
  B, T, C = x.size()                                    # x = (Batch,Time,Channe
  xx = self.time_shift(x)                               # Time Mix 部分介绍的 Tim
  xk = x * self.time_mix_k + xx * (1 - self.time_mix_k) # self.time_mix_k 是可学
  xv = x * self.time_mix_v + xx * (1 - self.time_mix_v) # self.time_mix_v 是可学
  xr = x * self.time_mix_r + xx * (1 - self.time_mix_r) # self.time_mix_r 是可学
  k = self.Linear_k(xk)                                 # self.Linear_k 是一个线
  v = self.Linear_v(xv)                                 # self.Linear_v 是一个线
  r = self.Linear_r(xr)                                 # self.Linear_r 是一个线
  sr = torch.sigmoid(r)

  k = torch.exp(k)                                      # k = (Batch,Time,Channe
  sum_k = torch.cumsum(k, dim=1)                        # sum_k = (Batch, Channe
  kv = (k * v).view(B, T, self.n_head, self.head_size)
  wkv = (torch.einsum('htu,buhc->bthc', w, kv)).contiguous().view(B, T, -1)
  rwkv = sr * wkv / sum_k
  return self.Linear_o(rwkv)                            # self.Linear_o 是一个线
```

Channel Mix

这一部分的改进原理相对简单，本质上就是使用了更加复杂的操作代替了简单的若干线性层。不过，在这个模块中，各个版本的 RWKV 设计稍有不同，我们参考了最新的 RWKV-v4 版本进行介绍，整体结构如下图所示：



由于这部分的代码较为简单直接，我们直接使用 PyTorch 风格的伪代码进行说明：

```
def forward(self, x):
  xx = self.time_shift(x)                                  # Time Mix 部分介绍的 Ti
  xk = x * self.time_mix_k + xx * (1 - self.time_mix_k)    # self.time_mix_k 是可学
  xr = x * self.time_mix_r + xx * (1 - self.time_mix_r)    # self.time_mix_r 是可学
  k = self.Linear_k(xk)                                    # self.Linear_k 是一个线
  k = torch.square(torch.relu(k))
  kv = self.Linear_v(k)                                    # self.Linear_v 是一个线
  return torch.sigmoid(self.Linear_r(xr)) * kv             # self.Linear_r 是一个线
```

可以看出，这个模块的所有乘法都是张量的逐元素相乘，因此保持了在通道维度进行信息交流的设计准则，并且设计更为复杂，实验效果也更好。

综上所述，在提出了 Channel Mix 模块和 Time Mix 模块之后，RWKV 不仅降低了 attention 的计算复杂度，同时保持了对数据的拟合能力，在运行速度上也有较大的提高，表现出了相当的潜力。