Transformer位置编码学习具有连续动态模型

Xuanqing Liu[†], Hsiang-Fu Yu[‡], Inderjit Dhillon[§‡], Cho-Jui Hsieh[†]

[†] UCLA [§] UT Austin [‡] Amazon Inc.

xqliu@cs.ucla.edu rofu.yu@gmail.com

inderjit@cs.utexas.edu chohsieh@cs.ucla.edu

Abstract

本文提出一种新的学习方法来编码非递归模型的位置信息，如Transformer模型。与RNN和LSTM通过顺序加载输入标记而包含归纳偏差不同，非递归模型对位置不太敏感。主要原因是输入单元之间的位置信息没有内在编码，即模型是置换等价的;这个问题解释了为什么所有现有模型在输入端都伴随一个正弦编码/嵌入层。然而，这种解决方案有明显的局限性:正弦编码不够灵活，因为它是手动设计的，并且不包含任何可学习的参数，而位置嵌入限制了输入序列的最大长度。因此，需要设计一个新的位置层，其中包含可学习的参数，以适应不同的数据集和不同的架构。同时，我们也希望编码能够根据可变长度的输入进行推断。在所提出的解决方案中，借鉴了最近的神经ODE方法，可以被视为ResNet的通用连续版本。该模型能够对多种动态系统进行建模。利用这种动态系统对编码结果沿位置索引的演化进行建模，从而克服了现有方法的上述局限性。在各种神经机器翻译和语言理解任务上评估了新位置层，实验结果显示比基线有一致的改进。

1 简介

基于Transformer的模型[1, 2, 3, 4, 5, 6] 已成为建模序列数据的最有效方法之一可变长度。transformer已经显示出对许多自然界的广泛适用性语言处理(NLP)任务，例如语言建模[4]，神经机器翻译(NMT) [1]和语言了解[2]。与传统的基于循环的方法不同模型(如RNN或LSTM)，Transformer使用非递归but 自注意力神经架构建模元素之间的依赖关系在序列中不同的位置，这有助于更好的并行化使用现代硬件，缓解了梯度消失/爆炸问题在传统的循环模型中。

[7] 证明了自注意力结构的设计导致了置换等价函数族。因此，对于元素排序很重要的应用程序，如何正确编码位置信息对基于Transformer的模型至关重要。已经有很多尝试为Transformer编码位置信息。在Transformer的原始论文[1]中，通过调整一族预定义的正弦函数来为每个位置构建一组嵌入。然后将这些固定位置的嵌入添加到输入序列的单词嵌入中。为了以更数据驱动的方式进一步构建这些位置嵌入，许多最近的Transformer变体，如[2, 8]，在训练阶段将这些嵌入作为可学习的模型参数。这种数据驱动的方法是以限制固定的最大输入序列长度$L_{\max}$和额外$L_{\max} \times d$参数的计算/内存开销为代价的，其中$L_{\max}$在许多应用程序中通常设置为512,d是嵌入的维度。[9]提出一种相对位置表示方法，通过丢弃距离大于K的标记之间的交互来减少到$(2K+1)d$的参数数量。除了输入层之外，[10]和[5]表明向每一层注入位置信息会导致Transformer的性能更好。

理想的位置编码方法应满足以下3个特性。

1. 归纳(归纳):能够处理比训练时间内看到的任何序列都长的序列。

Table 1: 比较位置表示方法

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal [1]	✓	✗	✓
Embedding [2]	✗	✓	✗
Relative [9]	✗	✓	✓
This paper	✓	✓	✓

2. 数据驱动:位置编码应该可以从数据中学习。
3. 参数效率:编码引入的可训练参数的数量应该受到限制，以避免模型大小的增加，这可能会损害泛化。

在表1中，我们根据这三个属性总结了一些现有的位置编码方法。

本文提出了一种新的最小代价位置编码方法。其主要思想是将位置编码建模为一个连续的动态系统，因此我们只需要学习系统动力学，而不是独立学习每个位置的嵌入。通过这样做，所提出方法享受了两个世界的优点——恢复了归纳偏差，编码方法是自由可训练的，同时参数高效。为了能够用反向传播来训练这个动态系统，我们采用了连续神经网络的最新进展[11]，正式称为神经ODE。在一些生成式建模文献中，它也被称为自由形式流模型[12]，因此我们将我们的模型称为基于流的**TransformER (FLOATER)**。我们的贡献如下:

- 本文提出FLOATER，一种新的Transformer位置编码器，以数据驱动和参数高效的方式，通过连续的动态模型对位置信息进行建模。
- 由于使用了连续动态模型，FLOATER可以处理任意长度的序列。这个特性使推理更加灵活。
- 经过精心设计，我们的位置编码器与原始变压器兼容;也就是。，原始的Transformer可以被视为我们所提出的位置编码方法的一个特殊情况。因此，我们不仅能够用FLOATER从头开始训练Transformer模型，还可以将FLOATER插入到大多数现有的预训练Transformer模型中，如BERT、RoBERTa、等等。
- 在机器翻译、语言理解和问答等各种NLP任务中，与基线模型相比，取得了一致的改进。

2 背景及相关工作

2.1 Transformer位置编码的重要性

我们使用一个简化的自注意力序列编码器来说明Transformer中位置编码的重要性。无位置编码后，Transformer架构可以被视为N的堆栈阻止$B_n : n = 1, \ldots, N$包含一个自我关注的A_n和一个前馈层F_n。通过丢弃剩余连接和层归一化，简化的Transformer编码器的架构可以表示如下。

$$\text{Encode}(\boldsymbol{x}) = B_N \circ B_{N-1} \circ \cdots \circ B_1(\boldsymbol{x}), \tag{1}$$

$$B_n(\boldsymbol{x}) = F_n \circ A_n(\boldsymbol{x}), \tag{2}$$

$\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L]^\top \in \mathbb{R}^{L \times d}$, L在哪里序列的长度，d是单词嵌入的维度。$A_n(\cdot)$和$F_n(\cdot)$分别是n -th块$B_n(\cdot)$中的自关注层和前馈层。

$A_1(\boldsymbol{x})$的每一行可以看作是值矩阵$\boldsymbol{V} \in \mathbb{R}^{L\times d}$的加权和，其权重由相似度决定密钥矩阵$\boldsymbol{K} \in \mathbb{R}^{L\times d}$与查询矩阵$\boldsymbol{Q} \in \mathbb{R}^{L\times d}$的得分如下:

$$
\begin{aligned}
&A_1(\boldsymbol{x}) = \texttt{Softmax}\Big(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\Big)\boldsymbol{V}, \\
&\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_L]^\top, \qquad \boldsymbol{q}_i = \boldsymbol{W}_q\boldsymbol{x}_i + \boldsymbol{b}_q, \\
&\boldsymbol{K} = [\boldsymbol{k}_1, \boldsymbol{k}_2, ..., \boldsymbol{k}_L]^\top, \qquad \boldsymbol{k}_i = \boldsymbol{W}_k\boldsymbol{x}_i + \boldsymbol{b}_k, \\
&\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_L]^\top, \qquad \boldsymbol{v}_i = \boldsymbol{W}_v\boldsymbol{x}_i + \boldsymbol{b}_v,
\end{aligned}
\tag{3}
$$

$\boldsymbol{W}_{q/k/v}$和$\boldsymbol{b}_{q/k/v}$是自注意力函数$A_1(\cdot)$中引入的权重和偏差参数。转换器中使用的前馈函数$F_1(\cdot)$的输出也是一个L行的矩阵。其中，i -th行的获取方式如下。

$$
\text{the } i\text{-th row of } F_1(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x}_i + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{4}
$$

其中$\boldsymbol{W}_{1,2}$和$\boldsymbol{b}_{1,2}$是线性变换的权重和偏差，$\sigma(\cdot)$是激活函数。从(3)和(4)不难看出，$A_1(\cdot)$和$F_1(\cdot)$都是排列等价的。因此，我们可以得出结论，在(1)中定义的整个函数也是置换等价的，即$\Pi \times \text{Encode}(\boldsymbol{x}) = \text{Encode}\,(\Pi \times \boldsymbol{x})$对于任何$L \times L$置换矩阵$\Pi$。这种置换等价性限制了在没有位置信息的情况下，Transformer对元素顺序很重要的序列进行建模。

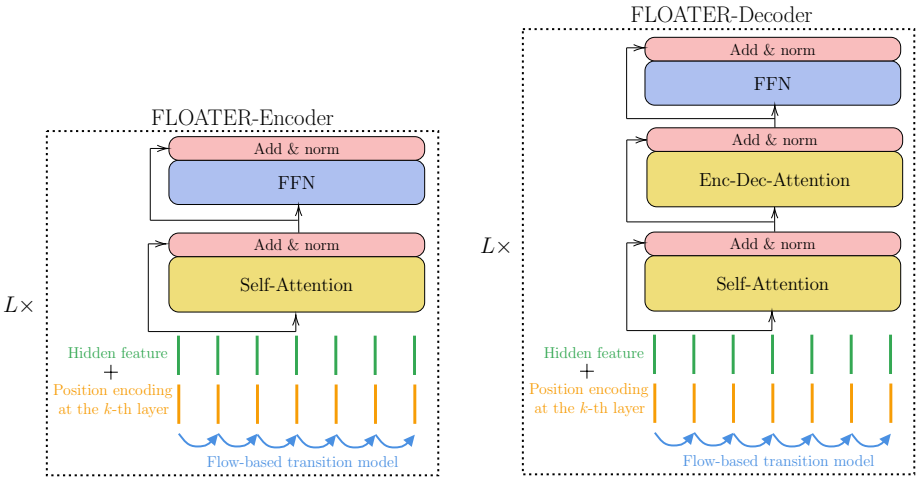

Figure 1: 我们的模型(FLOATER)的架构。FLOATER和原始Transformer模型之间的主要区别在于:1)位置表示被集成到层次结构中的每个块中(总共有N块);2)有一个动态模型(参见(8))，它为每个块生成位置编码向量。动力学是用一个黑盒微分方程求解器在补充材料详细。

2.2 Transformer中的位置编码

正如在1节中提到的，有许多尝试在自注意力组件中注入位置信息。它们中的大多数可以用下面的形式描述:

$$
B_n(\boldsymbol{x}) = F_n \circ A_n \circ \Phi_n(\boldsymbol{x}),\ n \in \{1, ..., N\}, \tag{5}
$$

其中$\Phi_n(\boldsymbol{x})$是位置编码函数。

[1] 建议保留$\Phi_n(\boldsymbol{x}) = \boldsymbol{x}, \forall n \geq 2$并仅在输入块上用预定义的正弦函数族注入位置信息:$\Phi_1(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(1)}$，其中$\boldsymbol{p}^{(1)} = [\boldsymbol{p}_1^{(1)}, \boldsymbol{p}_2^{(1)}, ..., \boldsymbol{p}_L^{(1)}]$是一个位置嵌入矩阵，$i$ -第一行对应于输入序列中的i -第th位置。具体来说，i -th行的j -th维定义如下。

$$\boldsymbol{p}_i^{(1)}[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) & \text{if } j \text{ is even,} \\ \cos(i \cdot c^{\frac{j-1}{d}}) & \text{if } j \text{ is odd,} \end{cases} \tag{6}$$

其中$c = 10^{-4}$。[10]和[5]通过进一步注入每个区块的位置信息来观察更好的性能，即$\Phi_n(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$，如下所示:

$$\boldsymbol{p}_i^{(n)}[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) + \sin(n \cdot c^{\frac{j}{d}}) & \text{if } j \text{ is even,} \\ \cos(i \cdot c^{\frac{j-1}{d}}) + \cos(n \cdot c^{\frac{j-1}{d}}) & \text{if } j \text{ is odd.} \end{cases} \tag{7}$$

注意，对于上述两种方法，所有应用程序的位置编码函数$\Phi_n(\cdot)$都是固定的。虽然在模型中没有引入额外的参数，但这两种方法都是归纳的，可以处理变长输入序列。
预训练Transformer模型的许多成功变体，如BERT [2]和RoBERTa [8]，都将$\Phi_1(\boldsymbol{x})$中的整个嵌入矩阵$\boldsymbol{p}^{(1)} \in \mathbb{R}^{L \times d}$作为训练参数。由于需要固定训练参数的数量，因此需要在训练前确定序列的最大长度L_{max}。尽管缺乏归纳特性，但这种数据驱动的方法被发现对许多NLP任务是有效的。请注意，与固定的正弦位置编码不同，由于有大量额外参数($NL_{\text{max}}d$)，没有试图在Transformer的每个块中注入一个可学习的位置嵌入矩阵。

3 FLOATER:我们提出的位置编码器

我们分三步介绍我们的方法。在第一步中，只研究一个变压器块，并描述如何学习由动力系统驱动的位置表示;在第二步中，如果我们向每一层添加位置信号，我们将展示如何保存参数;最后，我们略微更改了架构，以进一步保存可训练参数，并使FLOATER与原始Transformer“兼容”[1]。兼容性意味着我们的模型是普通Transformer的严格超集，因此可以从Transformer中初始化它。

3.1 基于动态系统的位置编码

Transformer模型中的位置表示是一系列向量$\{\boldsymbol{p}_i \in \mathbb{R}^d : i = 1, ..., L\}$被添加到序列中的输入表示形式$\{\boldsymbol{x}_i : i = 1, ..., L\}$。现有的位置编码方法要么应用一个固定的正弦函数来获得$\{\boldsymbol{p}_i\}$，要么包括它们作为不相关的可学习参数。它们都无法捕捉这些位置表示之间的依赖性或动态性$\{\boldsymbol{p}_i\}$。本文建议使用一个动态系统对这些位置表示进行建模;也就是说，从$\boldsymbol{p}_i$到$\boldsymbol{p}_{i+1}$，有一种“潜在的力量”，用$\boldsymbol{h}_i$表示。为了促进流畅性，我们将$\boldsymbol{p}(t) : \mathbb{R}_+ \mapsto \mathbb{R}^d$视为离散序列$\{\boldsymbol{p}_i\}$。特别地，我们提出的连续动力系统的特征为:

$$\boldsymbol{p}(t) = \boldsymbol{p}(s) + \int_s^t \boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)\, \mathrm{d}\tau,\ 0 \leq s \leq t < \infty, \tag{8}$$

以及初始向量$\boldsymbol{p}(0)$，其中$\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)$是一个神经网络参数为$\boldsymbol{\theta}_h$，并接受前一个状态$(\tau, \boldsymbol{p}(\tau))$。注意这个定义域$\boldsymbol{p}(\cdot)$就是$\mathbb{R}_+$。位置序列$\{\boldsymbol{p}_i\}$可以通过在一系列点$\{t_i : 0 \leq t_1 < \cdots \leq t_L\}$: $\boldsymbol{p}_i = \boldsymbol{p}(t_i)$上取$\boldsymbol{p}(\cdot)$得到。一个简单的策略是设置$t_i = i \cdot \Delta t$使这些点是等距的，其中$\Delta$是一个超参数(例如，$\Delta = 0.1$)。使用这种策略，我们隐含地假设，随着我们遍历句子中的每

个token，位置信号稳步发展。一般来说，$\{t_i\}$可以是任何单调递增的序列，这允许我们将我们的工作扩展到更多的应用，其中序列中的元素并不总是以相同的间隔观察。有关此一般设置的适用性的更多讨论包含在补充材料中。对于本文讨论的NLP应用，我们选择$t_i = i \cdot \Delta t$。Eq. (8)相当于一个ODE问题$\frac{\mathrm{d}\boldsymbol{p}(t)}{\mathrm{d}t} = \boldsymbol{h}(t, \boldsymbol{p}(t); \boldsymbol{\theta}_h)$，它保证在温和的条件下有一个唯一的解决方案[13]。我们遵循[11]的有效方法来计算$\boldsymbol{\theta}_h$相对于整体训练损失的梯度，这允许我们将这种参数化动态位置编码器包括到Transformer模型的端到端训练中。更多细节可以在补充材料中找到。

我们的动态系统(8)是相当灵活的，可以承认标准的正弦位置编码(6)作为一种特殊情况:

$$\begin{aligned}
&\boldsymbol{p}_{i+1}[j] - \boldsymbol{p}_i[j] \\
&= \begin{cases} \sin\left((i+1) \cdot c^{\frac{j}{d}}\right) - \sin\left(i \cdot c^{\frac{j}{d}}\right) & \text{if } j \text{ is even} \\ \cos\left((i+1) \cdot c^{\frac{j-1}{d}}\right) - \cos\left(i \cdot c^{\frac{j-1}{d}}\right) & \text{if } j \text{ is odd} \end{cases} \\
&= \begin{cases} \int_i^{i+1} c^{-\frac{j}{d}} \cos(\tau \cdot c^{\frac{j}{d}}) \, \mathrm{d}\tau & \text{if } j \text{ is even} \\ \int_i^{i+1} -c^{-\frac{j-1}{d}} \sin(\tau \cdot c^{\frac{j-1}{d}}) \, \mathrm{d}\tau & \text{if } j \text{ is odd,} \end{cases}
\end{aligned} \tag{9}$$

这表明对于简单正弦编码，存在一个同样是正弦函数的动态系统$\boldsymbol{h}(\cdot)$。

3.2 块之间的参数共享

如2节所述，在一些语言理解任务中，向Transformer的每个块注入位置信息可以带来更好的性能[10, 5]。我们提出的位置编码器浮动(8)也可以注入到每个块中。这个想法如图1所示。通常在sequence-to-sequence Transformer中有6块，在BERT中有12或24块。我们在n -th块上添加一个上标(n)来表示动态:

$$\boldsymbol{p}^{(n)}(t) = \boldsymbol{p}^{(n)}(s) + \int_s^t \boldsymbol{h}^{(n)}(\tau, \boldsymbol{p}^{(n)}(\tau); \boldsymbol{\theta}_h^{(n)}) \, \mathrm{d}\tau.$$

我们可以想象，为每个块使用N不同的动态模型$\boldsymbol{h}^{(n)}(\cdot; \boldsymbol{\theta}_h^{(n)})$会引入太多参数，并导致显著的训练开销。相反，我们通过在所有块中共享参数来解决这个问题，即

$$\boldsymbol{\theta}_h^{(1)} = \boldsymbol{\theta}_h^{(2)} = \cdots = \boldsymbol{\theta}_h^{(N)}. \tag{10}$$

请注意，(10)并不意味着所有$\boldsymbol{p}_t^{(n)}$都是相同的，因为我们将为每个块分配不同的初始值，即$\boldsymbol{p}^{(n_1)}(0) \neq \boldsymbol{p}^{(n_2)}(0)$为$n_1 \neq n_2$。

3.3 兼容性和热启动训练

在本节中，我们将改变添加位置编码的方式，以便可以直接从Transformer中初始化浮点数。例如，我们使用标准的Transformer模型，它在输入块具有固定的正弦编码，而在更深层没有位置编码。请注意，该技术可以扩展到具有不同位置编码方法的其他transformer变体，如嵌入矩阵。我们首先检查标准的Transformer模型，block- n上的查询矩阵$\boldsymbol{Q}^{(n)}$是

$$\widetilde{\boldsymbol{q}}_i^{(n)} = \boldsymbol{W}_q^{(n)}\left(\boldsymbol{x}_i + \widetilde{\boldsymbol{p}}_i^{(n)}\right) + \boldsymbol{b}_q^{(n)}, \tag{11}$$

其中$\boldsymbol{W}_q^{(n)}$和$\boldsymbol{b}_q^{(n)}$是A_n(3)中的参数;$\widetilde{\boldsymbol{p}}^{(n)}$是正弦编码;$\widetilde{\boldsymbol{q}}_i^{(n)}$是$\boldsymbol{Q}^{(n)}$的第$i$行。这里我们添加了一个波浪符号来表示正弦向量。$\widetilde{\boldsymbol{k}}_i^{(n)}$和$\widetilde{\boldsymbol{v}}_i^{(n)}$的公式具有非常相似的形式，为简洁起见省略。

	Transformer-Base		*Transformer-Large*	
	En-De	En-Fr	En-De	En-Fr
Position encoders at all blocks				
FLOATER	**28.6**	**41.6**	**29.2**	**42.7**
Pre-defined Sinusoidal Position Encoder	28.2	40.6	28.4	42.0
Fixed-length Position Embedding	26.9	40.9	28.3	42.0
Position encoder only at input block				
FLOATER	28.3	41.1	29.1	42.4
Pre-defined Sinusoidal Position Encoder	27.9	40.4	28.4	41.8
Fixed-length Position Embedding	27.8	40.9	28.5	42.4

Table 2: 不同位置编码器在机器翻译任务上的实验结果。

现在我们考虑浮动器的情况，其中添加了新的位置编码$\boldsymbol{p}_i$

$$\begin{aligned}\boldsymbol{q}_i^{(n)} &= \boldsymbol{W}_q^{(n)}\big(\boldsymbol{x}_i+\boldsymbol{p}_i\big)+\boldsymbol{b}_q^{(n)}\\ &= \underbrace{\boldsymbol{W}_q^{(n)}(\boldsymbol{x}_i+\widetilde{\boldsymbol{p}}_i^{(n)})+\boldsymbol{b}_q^{(n)}}_{\text{Eq. (11)}}+\underbrace{\boldsymbol{W}_q^{(n)}(\boldsymbol{p}_i-\widetilde{\boldsymbol{p}}_i^{(n)})}_{\text{Extra bias term depends on } i}\\ &= \widetilde{\boldsymbol{q}}_i^{(n)}+\boldsymbol{b}_{q,i}^{(n)}.\end{aligned} \tag{12}$$

很容易看出，将位置嵌入从$\{\widetilde{\boldsymbol{p}}_i^{(n)}\}$更改为$\{\boldsymbol{p}_i^{(n)}\}$相当于在每个自注意力层$\{A_n(\cdot)\}$中添加位置感知偏差向量$\boldsymbol{b}_{q,i}^{(n)}$。因此，我们可以使用(8)来模拟$\boldsymbol{b}_q^{(n)}$的动态。特别地，我们有以下动态系统:

$$\boldsymbol{b}_q^{(n)}(t)=\boldsymbol{b}_q^{(n)}(0)+\int_0^t \boldsymbol{h}^{(n)}(\tau,\boldsymbol{b}_q^{(n)}(\tau);\boldsymbol{\theta}_h)\,\mathrm{d}\tau. \tag{13}$$

之后，我们设置$\boldsymbol{b}_{q,i}^{(n)}=\boldsymbol{b}_q^{(n)}(i\cdot\Delta t)$。我们可以看到如果$\boldsymbol{h}(\cdot)=0$和$\boldsymbol{b}_q^{(n)}(0)=0$，那么$\boldsymbol{b}_q^{(n)}\equiv 0$。这意味着(12)退化为(11)。请注意，(13)与(8)具有相同的形式，只是我们现在在(3)中对偏差术语$\boldsymbol{b}_{q,i}$进行建模。我们将对$\boldsymbol{K}$和$\boldsymbol{V}$应用相同的技术。

总而言之，我们的模型与原始Transformer有紧密的联系:如果我们将所有动态模型设置为零，这意味着$\boldsymbol{h}(\tau,\boldsymbol{p}(\tau);\boldsymbol{\theta}_h)\equiv 0$，那么我们的浮点模型将等同于具有正弦编码的原始Transformer。同样的技巧也适用于位置嵌入的Transformer，如BERT [2]。

我们努力使我们的模型与原始Transformer兼容，原因如下。首先，原始的Transformer训练速度更快，因为它不包含任何循环计算;这与我们的动态模型(8)形成对比，在该模型中，下一个位置$\boldsymbol{p}_{i+1}$取决于前一个位置$\boldsymbol{p}_i$。通过利用模型架构的兼容性，可以直接从预训练的Transformer模型检查点初始化浮动模型，然后对下游任务进行更多epoch的微调。通过这样做，我们可以享受我们的FLOATER模型的所有好处，但仍然保持一个可接受的培训预算。同样，对于像BERT或Transformer-XL这样的模型，我们已经为下游任务提供了组织良好的检查点。这些模型从头开始训练的成本很高，由于我们的目标是检查我们提出的位置表示方法是否可以比原始方法改进，我们决定逐层复制注意力权重以及FFN层，并随机初始化动态模型$\boldsymbol{h}(\tau,\boldsymbol{p}(\tau);\boldsymbol{\theta}_h)$。

4 实验结果

在本节中，我们进行了实验，看看FLOATER是否可以在各种NLP任务上改进给定Transformer模型的现有位置编码方法。因此，本文报告的所有指标都是从每个评估NLP任务上的单个(而不是集成)Transformer模型计算的。尽管低于排行榜上的最高分数，但这些指标能够揭示更清晰的信号，以判断所提出位置编码器的有效性。
我们在本文中执行实验的所有代码都基于`fairseq`[14]包中的Transformer实现。实施细节可在补充材料中找到。我们的实验代码将公开。

Table 3: GLUE基准上的实验结果

Model	**Single Sentence**		**Similarity and Paraphrase**			**Natural Language Inference**		
	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE
Base model								
RoBERTa	**63.6**	94.8	88.2	**91.9**	91.2	87.6	92.8	78.7
FLOATER	63.4	**95.1**	**89.0**	91.7	**91.5**	**87.7**	**93.1**	**80.5**
Large model								
RoBERTa	68.0	96.4	90.9	**92.2**	92.4	90.2	94.7	86.6
FLOATER	**69.0**	**96.7**	**91.4**	**92.2**	**92.5**	**90.4**	**94.8**	**87.0**

Table 4: 在竞赛基准上的实验结果。' Middle'是指初中英语考试，' High'是指高中考试。其他细节可以在[15]找到。

Model	Accuracy	Middle	High
Single model on test, large model			
RoBERTa	82.8	86.5	81.3
FLOATER	**83.3**	**87.1**	**81.7**

4.1 神经机器翻译

神经机器翻译(NMT)是第一个证明序列到序列Transformer模型优于传统循环序列模型的应用。我们包括以下三个加性位置编码器:$\Phi^{(n)}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$。

- 数据驱动的浮动器:$\boldsymbol{p}^{(n)}$是由我们提出的具有(8)中描述的数据驱动参数的连续动态模型生成的。
- 预定义正弦位置编码器:$\boldsymbol{p}^{(n)}$由(7)中描述的预定义函数构造而成，该函数由[1]提出并由[10]扩展。
- 固定长度位置嵌入:$\boldsymbol{p}^{(n)}$被包含为可学习的训练参数。这是由[1]首次引入的，并在Transformer的许多变体[2, 8]中采用。

为了更好地演示FLOATER带来的参数效率，对于上述每个编码器，我们还包括两个实验设置:将编码器放置在所有块或仅在输入块(即$\boldsymbol{p}^{(n)} = 0, \forall n \geq 2$)。

在表2中，我们展示了WMT14 Ee-De和En-Fr数据集上的BLEU分数，其中包括[1]中描述的Transformer-*base*和Transformer-*large*模型。在所有数据/模型组合中，所提出的所有块上的浮动器优于其他两个位置编码器。
另一方面，在所有块上添加位置编码器比仅在输入块上添加位置编码器产生了更好的性能。虽然在固定长度位置嵌入方法中有一个例外。我们怀疑这种现象是由于这种方法引入的$L_{\max}dN$可学习参数导致的过拟合。相比之下，所提出的浮动器是参数高效的(更多讨论请参见4.3节)，因此可以通过在Transformer的所有块中注入位置编码器来提高性能，而不需要太多额外的开销。

4.2 语言理解和问答

Table 5: 在SQuAD基准上的实验结果。所有结果均来自RoBERTa-large模型。

Model	**SQuAD 1.1**		**SQuAD 2.0**	
	EM	F1	EM	F1
Single models on dev, w/o data augmentation				
RoBERTa	88.9	94.6	86.5	89.4
FLOATER	88.9	94.6	**86.6**	**89.5**

BERT和RoBERTa等预训练Transformer模型已经成为在各种语言理解和问答任务中取得最先进性能的关键。在本节中，我们希望评估所提出的浮动器在这些任务上的有效性。我们特别关注三个语言理解基准集GLUE [16], RACE [15]和SQuAD [17]。如3.3节所述，FLOATER经过精心设计，以与现有的Transformer模型兼容。因此，可以利用预训练的Transformer模型来热启动一个浮动模型，该模型很容易用于对这些NLP任务进行微调。本文从官方存储库下载了与本节讨论的所有*NLP*任务的预训练*Transformer*模型相同的预训练RoBERTa模型。胶水基准。 该基准通常用于评估NLP模型的语言理解能力。表3中的实验结果表明，我们的FLOATER模型在大多数数据集上都优于RoBERTa，即使唯一的区别是位置编码的选择。比赛基准与**GLUE**基准 类似，RACE基准是另一种广泛使用的语言理解测试套件。与GLUE相比，RACE中的每一项都包含了更长的上下文，我们认为这对掌握准确的位置信息有更重要的要求。像GLUE基准一样，我们从相同的预训练RoBERTa检查点对模型进行微调。我们保持超参数，如批量大小和学习率，也保持相同。表4显示了实验结果。我们再次看到，FLOATER在所有子任务上的性能都得到了一致的改善。

Table 6: 在WMT14 En-De数据和Transformer-base架构上的性能比较。BLUE分数和每个位置编码器内的可训练参数数量都包括在内。

	BLEU (↑)	#Parameters (↓)
FLOATER	28.57	526.3K
1-layer RNN + scalar	27.99	263.2K
2-layer RNN + scalar	28.16	526.3K
1-layer RNN + vector	27.99	1,050.0K

阵容基准 SQuAD基准[17, 18]是评估NLP模型问答技能的另一项具有挑战性的任务。在这个数

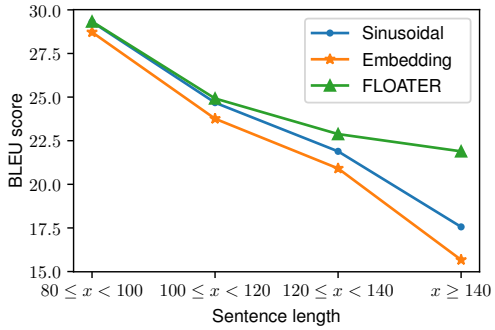

Figure 2: 比较不同编码方法的BLEU分数。

据集中，每个条目都包含一个很长的段落，其中包含与该段相关的事实和几个问题。模型需要预测回答问题的字符范围。在SQuAD-v2中，问题变得更具挑战性，问题可能无法被上下文回答。为了公平比较，我们遵循与BERT/RoBERTa相同的数据处理脚本;更多关于培训过程的细节在补充材料中描述。实验结果如表5所示。正如我们所看到的，FLOATER模型在大多数数据集上的表现都优于基准RoBERTa模型。考虑到两个模型都是从相同的预训练检查点进行微调的，改进是显著的。

4.3 更多讨论和分析

FLOATER是如何归纳的**?** FLOATER被设计成一个数据驱动的动态模型(8)。为了了解归纳浮动器与现有方法的比较情况，我们设计了以下实验。我们首先注意到，在WMT14 En-De数据集中，98.6%的训练句子比80 token短。基于此，我们创建了一个新的数据集，称为*En-De*由短到长(为简洁起见，或S2L):该数据集将所有短句($<$ 80 token)作为训练分割，所有长句子($\geq$ 80 token)作为测试分割。我们进一步根据$[80, 100)$, $[100, 120)$, $[120, 140)$, $[140, +\infty)$中下降的源长度将测试划分为四个箱子。BLEU分数在每个箱子中计算，结果如图2所示。
所提出的FLOATER模型在长句子上的表现特别好，即使在训练期间模型只看到了短句。这一经验观察支持了我们的猜想，即浮动模型是归纳的:从较短序列中学习到的动力学可以适当地推广到较长序列。

RNN是动态建模的一个很好的替代方案吗**?** 循环神经网络(Recurrent neural network, RNN)通常用于进行顺序建模。RNN和我们的连续动态模型(8)确实有一些共同点。i -th步骤的计算结果依赖于$(i-1)$ -st步骤的结果。此外，它们都包含可训练的参数，使它们能够适应每个特定的任务。最后，它们可以根据需要外推到任何长度。为了看看RNN是否同样有效，我们使用RNN模型对序列进行建模$\{\boldsymbol{p}_i\}_{i\in\{1,2,\dots\}}$:

$$\boldsymbol{p}_{i+1} = \mathrm{RNN}(\boldsymbol{z}_i, \boldsymbol{p}_i), \tag{14}$$

其中$\boldsymbol{z}_i \in \mathbb{R}^{d_{\text{in}}}$是索引$i$处RNN模型的输入。回想一下，在RNN语言模型中，$\boldsymbol{z}_i$是i -th标记的词嵌入或隐藏特征。在我们的例子中，由于我们应用RNN来学习编码而不是隐藏特征，合理的输入可以是标量值i或正弦编码的向量化值Vectorize(i)。我们在WMT14 En-De数据上尝试了两种选择，发现向量化的值通常效果更好，尽管不如我们的浮动模型。具体结果见表6。

每个位置编码是什么样子的**?** 为了更好地理解不同位置编码如何影响序列建模，在图3中，我们可视化了从WMT14 En-De数据集上的transformer基骨干的四种不同位置编码方法获得的

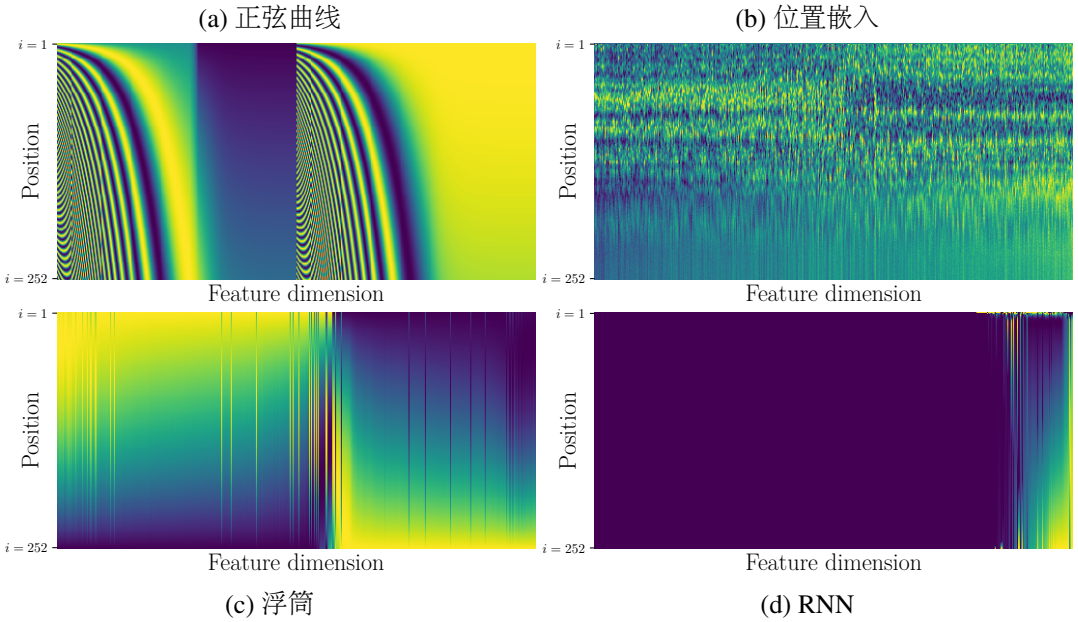

Figure 3: 可视化四种不同的定位方法。所有模型都使用Transformer-base架构和En-De数据集进行训练。为了更好的可视化，维度索引按顺序排列在图3b - 3d中。

位置嵌入矩阵$\boldsymbol{p}$。我们可以看到，正弦编码(3a)是最结构化的，而位置嵌入(3b)相当混乱。我们的FLOATER模型完全从数据中学习位置表示，但仍然表现出一些规律(3c)。最后，RNN模型(3d)无法提取足够的位置信息，可能是由于梯度消失问题。另一个发现是通过观察(3b)，我们观察到向量在不同的大位置中几乎是恒定的(在图3b的底部附近，我们看到具有相同颜色的垂直线的模式)。这种现象是由于数据集中的长句子很少，因此较低索引携带的位置信息无法外推到较高索引。相反，本文提出的动态模型兼有两方面的优点——它对数据集分布具有自适应能力，并且能够归纳处理长度大于训练集分割的序列。

4.4 关于培训和测试效率的评论

在训练期间，基于流的方法增加了不可忽略的时间和内存开销，这并不奇怪;这是因为精确求解神经ODE涉及流量模型的前向和后向传播∼100次。即使我们故意设计了一个小流模型(仅由两个FFN和一个非线性层组成)，将它们堆叠在一起仍然大大增加了训练时间。为了能够训练大型模型，我们使用以下优化:

- 使用不包含基于流的动态的预训练模型初始化，如3.3节所述。
- 从(8)，我们知道如果$\boldsymbol{h}(\cdot)$接近于零，那么位置信息减少(派生于附录)。这样，模型退化为原始的Transformer。受此特性的启发，我们可以用较小的权重初始化浮点数。结合上一个技巧，我们获得了一个有信息的初始化，在开始时产生了更低的训练损失。
- 我们观察到$\boldsymbol{h}(\cdot)$中的权重更稳定，更容易训练。因此，我们可以将$\boldsymbol{h}(\cdot)$的权重与Transformer模型的其余部分分开。具体来说，我们可以1)缓存位置偏差向量进行一些迭代，而无需重新计算;2)更新流模型权重的频率低于Transformer的其他部分;3)以较大的学习率更新流模型，以加快收敛。
- 对于RoBERTa模型，采用了一种更直接的策略:首先下载一个预训练的RoBERTa模型，插入一些基于流的编码层，并在WikiText-103数据集上重新训练一个epoch的编码层。在

对GLUE数据集进行微调时，我们可以选择冻结编码层。
结合这些技巧，与传统模型相比，仅用20 - 30 %的开销成功地训练了所提出的模型，并且在对GLUE基准进行微调RoBERTa模型时几乎没有开销。此外，如果将预先计算的位置偏差向量存储在检查点中，则在推理阶段没有任何开销。

5 结论

本文表明，用动态模型学习位置编码可以是改进Transformer模型的有利方法。所提出的位置编码方法是归纳的、数据驱动的和参数高效的。在各种自然语言处理任务(如神经机器翻译、语言理解和问答任务)上，证明了所提出模型比现有位置编码方法的优越性。

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[4] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[5] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[7] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[9] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, 2018.

[10] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[11] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[12] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[13] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. Dover Books on Mathematics. Dover Publications, 1985.

[14] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[15] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[16] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[17] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[18] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[19] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. Numerical recipes in c++. *The art of scientific computing*, 2:1002, 1992.

[20] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

[21] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[22] Yang Liu and Mirella Lapata. Hierarchical transformers for multi-document summarization. *arXiv preprint arXiv:1905.13164*, 2019.

[23] Xingxing Zhang, Furu Wei, and Ming Zhou. HIBERT: document level pre-training of hierarchical bidirectional transformers for document summarization. *CoRR*, abs/1905.06566, 2019.

[24] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018.

A 在Transformer中训练神经ODE模型

我们讨论了训练动态模型的细节$\boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)$，回想一下在我们的花朵模型中，函数$\boldsymbol{h}$通过生成位置编码向量序列$\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$隐式地加入计算图，以自由初始化向量$\boldsymbol{p}_0$为条件。生成步骤的迭代计算如下(假设我们选择两个连续标记之间的间隔为Δ)

$$\begin{aligned}
\boldsymbol{p}_1 &= \boldsymbol{p}_0 + \int_0^{\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau, \\
\boldsymbol{p}_2 &= \boldsymbol{p}_1 + \int_{\Delta}^{2\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau, \\
&\vdots \\
\boldsymbol{p}_N &= \boldsymbol{p}_{N-1} + \int_{(N-1)\Delta}^{N\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau.
\end{aligned} \tag{15}$$

最后，这个序列的损失L将是所有位置编码结果的函数$L = L(\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_N)$，这进一步是模型参数的函数$\boldsymbol{w}_h$。问题是如何通过反向传播计算梯度$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h}$。这个问题完全解决了神经ODE方法[11]与一个有效的伴随ODE求解器。为了说明这一原理，我们在图4中绘制了一个图表，展示了正向和反向传播。

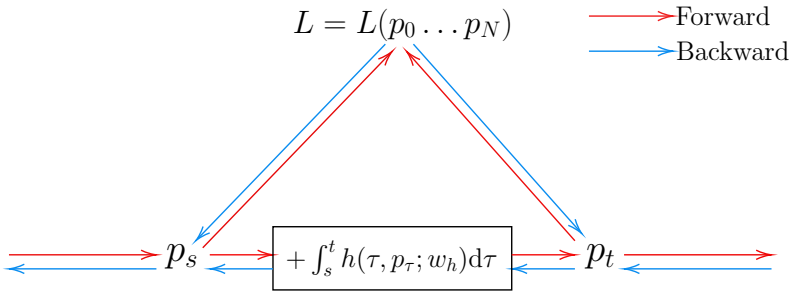

Figure 4: 正向和反向传播的方向。这里我们考虑一个简化版本，其中在计算图中只有位置编码$\boldsymbol{p}_s$和$\boldsymbol{p}_t$。

从[11]，我们知道梯度$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}_h} L\left(\boldsymbol{p}_s + \int_s^t \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau\right) \triangleq \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h}$可以计算为

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h} = -\int_t^s \boldsymbol{a}(\tau)^\intercal \frac{\partial \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h}\,\mathrm{d}\tau, \tag{16}$$

其中$\boldsymbol{a}(\tau)$定义在$\tau \in [s, t]$称为ODE的"伴随状态"，这可以通过解决另一个ODE

$$\frac{\mathrm{d}\boldsymbol{a}(\tau)}{\mathrm{d}\tau} = -\boldsymbol{a}(\tau)^\intercal \frac{\partial \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)}{\partial \boldsymbol{p}_\tau}. \tag{17}$$

请注意，(17)的计算只涉及雅可比向量积，因此可以通过自动微分有效地计算。

B 实现细节

B.1 ODE求解器的设置

要设置ODE服务器，我们需要首先选择数值算法[19]。对于不同的数据集，我们有不同的设置。对于神经机器翻译问题(WMT14 En-De和En-Fr)，我们使用具有离散步骤$\frac{\Delta}{5.0}$的更准确的龙格-库塔方案来解决伴随方程(回想一下，我们将两个相邻token的区间全局设置为$\Delta = 0.1$)。而对于GLUE和RACE等长句数据集，我们发现用高阶格式求解伴随方程太慢，在这种情况下，我们采用简单的中点法和离散步骤$\frac{\Delta}{5.0}$，并通过自动微分而不是伴随方法计算梯度。ODE求解器的第三方实现可以在`https://github.com/rtqichen/torchdiffeq`找到。

B.2 训练NMT任务

我们运行由fairseq [14]提供的相同的预处理脚本，它也用于ScalingNMT [20]。使用标准的训练脚本，我们首先成功重现了Transformer论文[1]中的所有结果。基于此，我们执行以下协议来获取结果:

1. 训练原始Transformer模型30个epoch。
2. 随机初始化相同形状配置的花朵模型。
3. 从性能最好的检查点(验证集)复制张量来初始化花朵模型。用小值初始化动态模型中的权重。
4. 峰值学习率的一半(例如，在Transformer-base + En-De中，峰值学习率从7.0×10^{-4}更改为3.5×10^{-4})。
5. 使用热初始化的花朵检查点，在同一数据集上重新训练10 epoch (En-De)或1 epoch (En-Fr)。
6. 平均最近5个检查点，并计算测试分割上的BLEU分数。

B.3 训练语言理解任务

对于GLUE/SQuAD/RACE基准，我们的实验都是在RoBERTa上进行的，其中基本配置和大型配置都可用。由于资源约束(并为了显示与现有模型的兼容性)，用预训练的RoBERTa初始化了花卉模型，这类似于NMT任务。然而，动态函数$\boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)$中的权重$\boldsymbol{w}_h$没有在大型语料库中进行训练，鉴于GLUE/SQuAD/RACE数据集太小，无法从头开始训练动态，我们决定使用掩码语言建模损失在WikiText103 [21]数据中单独预训练$\boldsymbol{h}$。我们发现，当我们单独训练$\boldsymbol{w}_h$时，只需要几个小时(2倍的Titan V100)和一个epoch就可以收敛。
一旦有了预训练的花模型，我们就可以运行以下下游任务，并在相同的设置下与RoBERTa进行比较:

胶水基准　由8个数据集组成，每个数据集都有不同的超参数设置。对于学习率、批量大小、训练迭代次数、预热迭代次数等超参数，我们使用RoBERTa官方存储库[1]推荐的相同值。

[1]载　于:　`https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.glue.md`

阵容基准。 对于这个基准测试，我们编写了自己的微调代码，因为目前没有官方代码可用。在实现过程中，主要参考了第三方存储库[2]。我们无法完全匹配RoBERTa论文中报告的官方结果，但相当接近(∼0.1差异在F1)。对于我们的FLOWER模型，我们使用与RoBERTa相同的超参数。

比赛基准。 该基准具有最长的上下文和序列长度。我们遵循官方训练脚本[3]并重现结果。与其他基准类似，然后我们使用完全相同的训练超参数重复训练过程，以进行公平的比较。在这个基准中，我们冻结了权重$\boldsymbol{w}_h$，只微调RoBERTa的权重。

C 适合非等距判别的情况

虽然我们的模型允许在(8)中使用s和t的连续值，将范围限制在文本建模任务中，但位置是作为$\{0, 1, 2, \dots\}$的离散值。一旦获得了位置表示的连续版本$\boldsymbol{p}_t$，我们只需将经过区分的$\{\boldsymbol{p}_0, \boldsymbol{p}_\Delta, \boldsymbol{p}_{2\Delta}, \dots, \}$作为输入Transformer模型的实际值，其中$\Delta$是一个超参数(例如$\Delta = 0.1$)。通过等距离地选择位置$t$，我们隐含地假设位置信号随着我们遍历句子中的每个标记而稳定地演变。更一般地，(8)中的动力学可以处理位置不是整数$0, 1, 2, \dots$等的情况，而是任意的非等距单调递增序列$t_0 < t_1 < t_2 < \dots$。在附录中，我们以几个广泛部署的任务举例说明这种情况。我们认为这是一个有趣的未来方向。这使得我们的模型特别适合以下场景，但传统的位置表示可能不擅长:

- 层次**Transformer**模型[22, 23]。该模型是分层递阶循环神经网络的直接扩展，常用于长文档处理。它的工作原理是首先在每个句子上运行一个单词级的Transformer模型以提取句子嵌入，然后应用一个句子级的Transformer依次扫描每个句子嵌入。我们认为，在句子级别处理时，最好设置位置索引的增量$t_{i+1} - t_i$与i -th句子的长度成正比。这是因为较长的句子往往包含更多的信息，因此$\boldsymbol{p}_{i+1}$可能会远离$\boldsymbol{p}_i$。
- 时间序列事件的**Transformer**。由于测量时间是连续的，因此时间序列数据是连续位置比离散位置更有意义的另一种情况。更重要的是，通过对不规则时间格点上的历史观测值进行建模来预测未来值时，最好考虑两个连续测量值之间的时间区间长度。之前一个成功的工作是潜ODE [24]，除了他们使用RNN作为主干，并且他们用神经ODE对隐藏状态而不是位置表示进行建模(因为RNN本身提供了位置偏差)。

在本文中，我们不打算探讨上面讨论的更一般的情况。相反，我们决定把它们作为未来有趣的工作。

[2]主 要 是https://github.com/ecchochan/roberta-squad和https://github.com/huggingface/transformers

[3]https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.race.md