Learning to Encode Position for Transformer with Continuous Dynamical Model

Xuanqing Liu [1] **Hsiang-Fu Yu** [2] **Inderjit S. Dhillon** [3 2] **Cho-Jui Hsieh** [1]

Abstract

We introduce a new way of learning to encode position information for non-recurrent models, such as Transformer models. Unlike RNN and LSTM, which contain inductive bias by loading the input tokens sequentially, non-recurrent models are less sensitive to position. The main reason is that position information among input units is not inherently encoded, i.e., the models are permutation equivalent; this problem justifies why all of the existing models are accompanied by a sinusoidal encoding/embedding layer at the input. However, this solution has clear limitations: the sinusoidal encoding is not flexible enough as it is manually designed and does not contain any learnable parameters, whereas the position embedding restricts the maximum length of input sequences. It is thus desirable to design a new position layer that contains learnable parameters to adjust to different datasets and different architectures. At the same time, we would also like the encodings to extrapolate in accordance with the variable length of inputs. In our proposed solution, we borrow from the recent Neural ODE approach, which may be viewed as a versatile continuous version of a ResNet. This model is capable of modeling many kinds of dynamical systems. We model the evolution of encoded results along position index by such a dynamical system, thereby overcoming the above limitations of existing methods. We evaluate our new position layers on a variety of neural machine translation and language understanding tasks, the experimental results show consistent improvements over the baselines.

[1]Department of Computer Science, University of California Los Angeles, CA, USA [2]Amazon.com [3]Department of Computer Science, University of Texas Austin, TA, USA. Correspondence to: Xuanqing Liu <xqliu@cs.ucla.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

1. Introduction

Transformer based models (Vaswani et al., 2017; Devlin et al., 2018; Yang et al., 2019; Radford et al., 2019; Lan et al., 2019; Raffel et al., 2019) have become one of the most effective approaches to model sequence data of variable lengths. Transformers have shown wide applicability to many natural language processing (NLP) tasks such as language modeling (Radford et al., 2019), neural machine translation (NMT) (Vaswani et al., 2017), and language understanding (Devlin et al., 2018). Unlike traditional recurrent-based models (e.g., RNN or LSTM), Transformer utilizes a non-recurrent but self-attentive neural architecture to model the dependency among elements at different positions in the sequence, which leads to better parallelization using modern hardware and alleviates the vanishing/exploding gradient problem in traditional recurrent models.

It is known that the self-attentive architecture corresponds to a family of permutation equivalence functions. Thus, for applications where the ordering of the elements matters, how to properly encode position information is crucial for Transformer based models. There have been many attempts to encode position information for the Transformer. In the original Transformer paper (Vaswani et al., 2017), a family of pre-defined sinusoidal functions was adapted to construct a set of embeddings for each position. These fixed position embeddings are then added to the word embeddings of the input sequence accordingly. To further construct these position embeddings in a more data-driven way, many recent Transformer variants such as (Devlin et al., 2018; Liu et al., 2019) include these embeddings as learnable model parameters in the training stage. This data-driven approach comes at the cost of limiting the maximum length of sequence $L_{\max}$ and the computational/memory overhead from additional $L_{\max} \times d$ parameters, where $L_{\max}$ is usually set to 512 in many applications, and d is the dimension of the embeddings. (Shaw et al., 2018) propose a relative position representation to reduce the number of parameters to $(2K+1)d$ by dropping the interactions between tokens with a distance greater than K. In addition to just the input layer, (Dehghani et al., 2018) and (Lan et al., 2019) suggest that the injection of position information to every layer leads to even better performance for the Transformer. More

recently, (Wang et al., 2019) proposes to encode positions by complex numbers of different phases and wave-lengths. Despite being successful in several tasks, it increases the embedding matrix size by a factor of three.

An ideal position encoding approach should satisfy the following three properties:

1. **Inductive**: the ability to handle sequences longer than any sequence seen in the training time.
2. **Data-Driven**: the position encoding should be learnable from the data.
3. **Parameter Efficient**: number of trainable parameters introduced by the encoding should be limited to avoid increased model size, which could hurt generalization.

In Table 1, we summarize some of the existing position encoding approaches in terms of these three properties.

In this paper, we propose a new method to encode position with minimum cost. The main idea is to model position encoding as a continuous dynamical system, so we only need to learn the system dynamics instead of learning the embeddings for each position independently. By doing so, our method enjoys the best of both worlds – we bring back the inductive bias, and the encoding method is freely trainable while being parameter efficient. To enable training of this dynamical system with backpropagation, we adopt the recent progress in continuous neural network (Chen et al., 2018), officially called Neural ODE. In some generative modeling literature, it is also called the free-form flow model (Grathwohl et al., 2018), so we call our model **FLOw-bAsed TransformER (FLOATER)**. We highlight our contributions as follows:

- We propose FLOATER, a new position encoder for Transformer, which models the position information via a continuous dynamical model in a data-driven and parameter-efficient manner.
- Due to the use of a continuous dynamic model, FLOATER can handle sequences of any length. This property makes inference more flexible.
- With careful design, our position encoder is **compatible** with the original Transformer; *i.e.*, the original Transformer can be regarded as a special case of our proposed position encoding approach. As a result, we are not only able to train a Transformer model with FLOATER from scratch but also plug FLOATER into most existing pre-trained Transformer models such as BERT, RoBERTa, *etc.*
- We demonstrate that FLOATER consistent improvements over baseline models across a variety of NLP tasks ranging from machine translations, language understanding, and question answering.

Table 1. Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal (Vaswani et al., 2017)	✓	✗	✓
Embedding (Devlin et al., 2018)	✗	✓	✗
Relative (Shaw et al., 2018)	✗	✓	✓
This paper	✓	✓	✓

2. Background and Related Work

2.1. Importance of Position Encoding for Transformer

We use a simplified self-attentive sequence encoder to illustrate the importance of position encoding in the Transformer. Without position encoding, the Transformer architecture can be viewed as a stack of N blocks $B_n : n = 1, \dots, N$ containing a self-attentive A_n and a feed-forward layer F_n. By dropping the residual connections and layer normalization, the architecture of a simplified Transformer encoder can be represented as follows.

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

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

where $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_L]^\top \in \mathbb{R}^{L\times d}$, L is the length of the sequence and d is the dimension of the word embedding. $A_n(\cdot)$ and $F_n(\cdot)$ are the self-attentive and feed-forward layer in the n-th block $B_n(\cdot)$, respectively.

Each row of $A_1(\boldsymbol{x})$ can be regarded as a weighted sum of the value matrix $\boldsymbol{V} \in \mathbb{R}^{L\times d}$, with the weights determined by similarity scores between the key matrix $\boldsymbol{K} \in \mathbb{R}^{L\times d}$ and query matrix $\boldsymbol{Q} \in \mathbb{R}^{L\times d}$ as follows:

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

$\boldsymbol{W}_{q/k/v}$ and $\boldsymbol{b}_{q/k/v}$ are the weight and bias parameters introduced in the self-attentive function $A_1(\cdot)$. The output of the feed-forward function $F_1(\cdot)$ used in the Transformer is also a matrix with L rows. In particular, the i-th row is obtained as follows.

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

where $\boldsymbol{W}_{1/2}$ and $\boldsymbol{b}_{1/2}$ are the weights and biases of linear transforms, and $\sigma(\cdot)$ is the activation function. It is not hard to see from (3) and (4) that both $A_1(\cdot)$ and $F_1(\cdot)$ are permutation equivalent. Thus, we can conclude that the entire function defined in (1) is also permutation equivalent, i.e., $\Pi \times \text{Encode}(\boldsymbol{x}) = \text{Encode}(\Pi \times \boldsymbol{x})$ for any $L \times L$ permutation matrix Π. This permutation equivalence property restricts the Transformer without position information from modeling sequences where the ordering of elements matters.

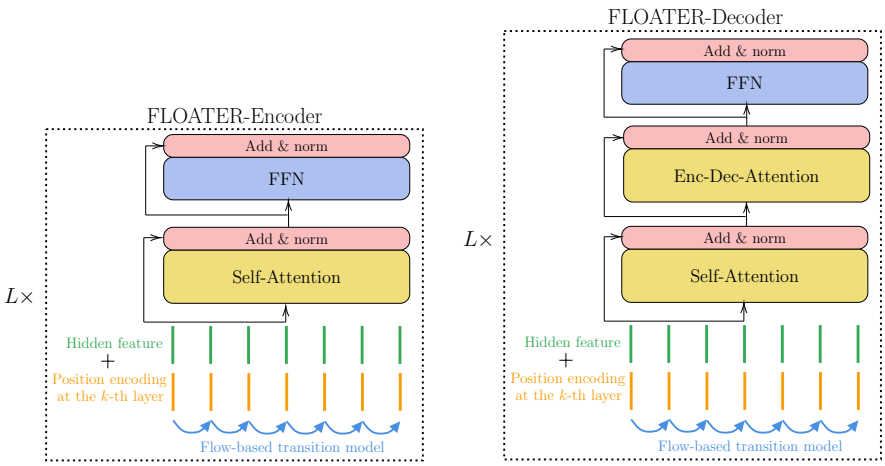

Figure 1. The architecture of our model (FLOATER). The main differences between FLOATER and the original Transformer model are: 1) the position representation is integrated into each block in the hierarchy (there are N blocks in total); and 2) there is a dynamical model (see (8)) that generates position encoding vectors for each block. The dynamics are solved with a black-box ODE solver detailed in the supplementary material.

2.2. Position Encoding in Transformer

As mentioned in Section 1, there are many attempts to inject position information in self-attentive components. Most of them can be described in the following form:

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

where $\Phi_n(\boldsymbol{x})$ is a position encoding function.

(Vaswani et al., 2017) propose to keep $\Phi_n(\boldsymbol{x}) = \boldsymbol{x}, \forall n \geq 2$ and inject position information only at the input block with a family of pre-defined sinusoidal functions: $\Phi_1(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(1)}$, where $\boldsymbol{p}^{(1)} = [\boldsymbol{p}_1^{(1)}, \boldsymbol{p}_2^{(1)}, ..., \boldsymbol{p}_L^{(1)}]$ is a position embedding matrix with the i-th row corresponding to the i-th position in the input sequence. In particular, the j-th dimension of the i-th row is defined as follows.

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

where $c = 10^{-4}$. (Dehghani et al., 2018) and (Lan et al., 2019) observe better performance by further injecting the position information at each block, i.e., $\Phi_n(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$ as follows:

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

Note that for the above two approaches, position encoding functions $\Phi_n(\cdot)$ are fixed for all the applications. Although no additional parameters are introduced in the model, both approaches are inductive and can handle input sequences of variable length.

Many successful variants of pre-trained Transformer models, such as BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019), include the entire embedding matrix $\boldsymbol{p}^{(1)} \in \mathbb{R}^{L \times d}$ in $\Phi_1(\boldsymbol{x})$ as training parameters. As the number of training parameters needs to be fixed, the maximum length of a sequence, L_{max}, is required to be determined before the training. Although it lacks the inductive property, this data-driven approach is found to be effective for many NLP tasks. Note that, unlike the fixed sinusoidal position encoding, there is no attempt to inject a learnable position embedding matrix at each block for Transformer due to a large number of additional parameters ($NL_{\text{max}}d$).

3. FLOATER: Our Proposed Position Encoder

We introduce our method in three steps. In the first step, we only look at one Transformer block, and describe how to learn the position representation driven by a dynamical system; in the second step, we show how to save parameters if we add position signals to every layer; lastly, we slightly change the architecture to save trainable parameters further and make FLOATER "compatible" with the original Transformer (Vaswani et al., 2017). The compatibility means our model is a strict superset of the vanilla Transformer so that it can be initialized from the Transformer.

3.1. Position Encoding with Dynamical Systems

Position representations in Transformer models are a sequence of vectors $\{\boldsymbol{p}_i \in \mathbb{R}^d : i = 1, ..., L\}$ to be added to the sequence of the input representations $\{\boldsymbol{x}_i : i = 1, ..., L\}$. Existing position encoding approaches either apply a fixed

sinusoidal function to obtain $\{\boldsymbol{p}_i\}$, or include them as uncorrelated learnable parameters. Both of them fail to capture the dependency or dynamics among these position representations $\{\boldsymbol{p}_i\}$. In this paper, we propose to use a dynamical system to model these position representations; that is, there is a "latent force" denoted by $\boldsymbol{h}_i$ that drives the changes from $\boldsymbol{p}_i$ to $\boldsymbol{p}_{i+1}$. To encourage smoothness, we consider $\boldsymbol{p}(t) : \mathbb{R}_+ \mapsto \mathbb{R}^d$ as the continuous version of the discrete sequence $\{\boldsymbol{p}_i\}$. In particular, our proposed continuous dynamical system is characterized as follows:

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

together with an initial vector $\boldsymbol{p}(0)$, where $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)$ is a neural network parameterized by $\boldsymbol{\theta}_h$ and takes the previous state $(\tau, \boldsymbol{p}(\tau))$. Notice that the domain of $\boldsymbol{p}(\cdot)$ is $\mathbb{R}_+$. The position sequence $\{\boldsymbol{p}_i\}$ can be obtained by taking $\boldsymbol{p}(\cdot)$ on a series of points $\{t_i : 0 \le t_1 < \cdots \le t_L\}$: $\boldsymbol{p}_i = \boldsymbol{p}(t_i)$. One simple strategy is to set $t_i = i \cdot \Delta t$ so that the points are equidistant, where Δ is a hyperparameter (e.g., $\Delta = 0.1$). With this strategy, we are implicitly assuming the position signals evolve steadily as we go through each token in a sentence. In general, $\{t_i\}$ can be any monotonically increasing series, which allows us to extend our work to more applications where the elements in the sequence are not always observed with the same interval. More discussions about the applicability for this general setting is included in the Supplementary material. For the NLP applications discussed in this paper, we choose $t_i = i \cdot \Delta t$.

Eq. (8) is equivalent to an ODE problem $\frac{\mathrm{d}\boldsymbol{p}(t)}{\mathrm{d}t} = \boldsymbol{h}(t, \boldsymbol{p}(t); \boldsymbol{\theta}_h)$, which is guaranteed to have a unique solution under mild conditions (Tenenbaum & Pollard, 1985). We follow the efficient approach by (Chen et al., 2018) to calculate the gradients of $\boldsymbol{\theta}_h$ with respect to the overall training loss, which allows us to include this parameterized dynamical position encoder into the end-to-end training of Transformer models. More details can be found in the Supplementary material.

Our dynamical system (8) is quite flexible to admit the standard sinusoidal position encoding (6) as a special case:

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

This indicates that for simple sinusoidal encoding, there exists a dynamical system $\boldsymbol{h}(\cdot)$ which is also sinusoidal function.

3.2. Parameter Sharing among Blocks

As mentioned in Section 2, injecting position information to each block for Transformer leads to better performance (Dehghani et al., 2018; Lan et al., 2019) in some language understanding tasks. Our proposed position encoder FLOATER (8) can also be injected into each block. The idea is illustrated in Figure 1. Typically there are 6 blocks in sequence-to-sequence Transformer and 12 or 24 blocks in BERT. We add a superscript (n) to denote dynamics at n-th block:

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

As we can imagine, having N different dynamical models $\boldsymbol{h}^{(n)}(\cdot; \boldsymbol{\theta}_h^{(n)})$ for each block can introduce too many parameters and cause significant training overhead. Instead, we address this issue by sharing parameters across all the blocks, namely

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

Note that (10) does not imply that all the $\boldsymbol{p}_t^{(n)}$ are the same, as we will assign different initial values for each block, that is $\boldsymbol{p}^{(n_1)}(0) \neq \boldsymbol{p}^{(n_2)}(0)$ for $n_1 \neq n_2$.

3.3. Compatibility and Warm-start Training

In this section, we change the way to add position encoding so that our FLOATER can be directly initialized from Transformer. As an example, we use the standard Transformer model, which has a fixed sinusoidal encoding at the input block and no position encoding at deeper levels. Note that this technique can be extended to other variants of Transformers with different position encoding methods, such as embedding matrix. We first examine the standard Transformer model, the query matrix $\boldsymbol{Q}^{(n)}$ at block-n is

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

where $\boldsymbol{W}_q^{(n)}$ and $\boldsymbol{b}_q^{(n)}$ are parameters in A_n (3); $\widetilde{\boldsymbol{p}}^{(n)}$ is the sinusoidal encoding; $\widetilde{\boldsymbol{q}}_i^{(n)}$ is the i-th row of $\boldsymbol{Q}^{(n)}$. Here we add a tilde sign to indicate the sinusoidal vectors. Formulas for $\widetilde{\boldsymbol{k}}_i^{(n)}$ and $\widetilde{\boldsymbol{v}}_i^{(n)}$ have a very similar form and are omitted for brevity.

Now we consider the case of FLOATER, where new position encodings $\boldsymbol{p}_i$ are added

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

	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. Experimental results of various position encoders on the machine translation task.

It is easy to see that the changing the position embedding from $\{\widetilde{\boldsymbol{p}}_i^{(n)}\}$ to $\{\boldsymbol{p}_i^{(n)}\}$ is equivalent to adding a position-aware bias vector $\boldsymbol{b}_{q,i}^{(n)}$ into each self-attentive layers $\{A_n(\cdot)\}$. As a result, we can instead apply (8) to model the dynamics of $\boldsymbol{b}_q^{(n)}$. In particular, we have the following dynamical system:

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

After that, we set $\boldsymbol{b}_{q,i}^{(n)} = \boldsymbol{b}_q^{(n)}(i \cdot \Delta t)$. We can see that if $\boldsymbol{h}(\cdot) = 0$ and $\boldsymbol{b}_q^{(n)}(0) = 0$, then $\boldsymbol{b}_q^{(n)} \equiv 0$. This implies (12) degenerates to (11). Note that (13) has the same form as (8), except that we are now modeling the bias terms $\boldsymbol{b}_{q,i}$ in (3). We will apply the same technique to $\boldsymbol{K}$ and $\boldsymbol{V}$.

To summarize, our model has a tight connection to the original Transformer: if we set all dynamical models to zero, which means $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h) \equiv 0$, then our FLOATER model will be equivalent to the original Transformer with the sinusoidal encoding. The same trick also works for Transformer with position embedding such as BERT (Devlin et al., 2018).

We strive to make our model compatible with the original Transformer due to the following reasons. First of all, the original Transformer is faster to train as it does not contain any recurrent computation; this is in contrast to our dynamical model (8), where the next position $\boldsymbol{p}_{i+1}$ depends on the previous one $\boldsymbol{p}_i$. By leveraging the compatibility of model architecture, we can directly initialize FLOATER model from a pre-trained Transformer model checkpoint and then fine-tune for the downstream task for a few more epochs. By doing so, we enjoy all the benefits of our FLOATER model but still maintain an acceptable training budget. Likewise, for models such as BERT or Transformer-XL, we already have well-organized checkpoints out of the box for downstream tasks. These models are costly to train from scratch, and since our goal is to examine whether our proposed position representation method can improve over the original one, we decided to copy the weights layer by layer for attention as well as FFN layers, and randomly initialize the dynamical model $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)$.

4. Experimental Results

In this section, we perform experiments to see if FLOATER can improve over the existing position encoding approaches for a given Transformer model on various NLP tasks. Thus, all the metrics reported in this paper are computed from a single (not ensemble) Transformer model over each evaluation NLP task. Albeit lower than top scores on the leaderboard, these metrics are able to reveal more clear signal to judge the effectiveness of the proposed position encoder.

All our codes to perform experiments in this paper are based on the Transformer implementations in the `fairseq` (Ott et al., 2019) package. Implementation details can be found in the Supplementary material. Our experimental codes will be made publicly available.

4.1. Neural Machine Translation

Neural Machine Translation (NMT) is the first application that demonstrates the superiority of a sequence-to-sequence Transformer model over conventional recurrent sequence models. We include the following three additive position encoders: $\Phi^{(n)}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$.

- **Data-driven FLOATER**: $\boldsymbol{p}^{(n)}$ is generated by our proposed continuous dynamical models with data-driven parameters described in (8).
- **Pre-defined sinusoidal position encoder**: $\boldsymbol{p}^{(n)}$ is constructed by a pre-defined function described in (7), which is proposed by (Vaswani et al., 2017) and extended by (Dehghani et al., 2018).
- **Length-fixed position embedding**: $\boldsymbol{p}^{(n)}$ is included as

Table 3. Experimental results on GLUE benchmark

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. Experiment results on RACE benchmark. "Middle" means middle school level English exams, "High" means high school exams. Other details can be found in (Lai et al., 2017).

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

Table 5. Experiment results on SQuAD benchmark. All results are obtained from RoBERTa-large model.

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**

learnable training parameters. This is first introduced by (Vaswani et al., 2017) and adopted in many variants of Transformer (Devlin et al., 2018; Liu et al., 2019).

To better demonstrate the parameter efficiency brought by FLOATER, for each above encoder, we also include two experimental settings: position encoder at all blocks or only at the input block (i.e., $\boldsymbol{p}^{(n)} = 0, \forall n \geq 2$).

In Table 2, we present the BLEU scores on WMT14 Ee-De and En-Fr datasets with both *Transformer-base* and *Transformer-large* models described in (Vaswani et al., 2017). Among all the data/model combinations, our proposed FLOATER at all blocks outperforms two other position encoders.

On the other hand, we also observe that adding position encoders at all blocks yields better performance than only at the input block. While there is an exception in the fixed-length position embedding approach. We suspect that this phenomenon is due to over-fitting cased by $L_{\max} dN$ learnable parameters introduced by this approach. In contrast, our proposed FLOATER is parameter efficient (more discussions in Section 4.3), so the performance can be improved by injecting the position encoder at all the blocks of Transformer without much additional overhead.

4.2. Language Understanding and Question Answering

Pretrained Transformer models such as BERT and RoBERTa have become the key to achieving the state-of-the-art performance for various language understanding and question answering tasks. In this section, we want to evaluate the effectiveness of the proposed FLOATER on these tasks. In particular, we focus on three language understanding benchmark sets, GLUE (Wang et al., 2018), RACE (Lai et al., 2017) and SQuAD (Rajpurkar et al., 2016). As mentioned in Section 3.3, FLOATER is carefully designed to be compatible with the existing Transformer models. Thus, we can utilize pretrained Transformer models to warm-start a FLOATER model easily to be used to finetune on these NLP tasks. In this paper, we download the *same* pre-trained RoBERTa model from the official repository as our pre-trained Transformer model for all NLP tasks discussed in this section.

GLUE Benchmark. This benchmark is commonly used to evaluate the language understanding skills of NLP models. Experimental results in Table 3 show that our FLOATER model outperforms RoBERTa in most datasets, even though the only difference is the choice of positional encoding.

RACE benchmark Similar to the GLUE benchmark, the RACE benchmark is another widely used test suit for language understanding. Compared with GLUE, each item in RACE contains a significantly longer context, which we believe requires more important to grasp the accurate position information. Like in GLUE benchmark, we finetune the model from the same pretrained RoBERTa checkpoint. We keep the hyperparameters, such as batch size and learning rate, to also be the same. Table 4 shows the experimental results. We again see consistent improvement of FLOATER across all subtasks.

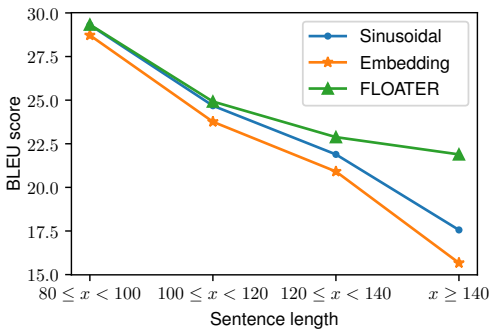

Figure 2. Comparing BLEU scores of different encoding methods.

Table 6. Performance comparison on WMT14 En-De data and Transformer-base architecture. Both BLUE scores and the number of trainable parameters inside each position encoder are included.

	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
1-Layer LSTM + scalar	28.17	1050.0K
1-Layer GRU + scalar	28.11	789.5K
2-Layer LSTM + scalar	28.10	2100.0K
2-Layer GRU + scalar	26.21	1580.0K

SQuAD benchmark SQuAD benchmark (Rajpurkar et al., 2016; 2018) is another challenging task to evaluate the question answering skills of NLP models. In this dataset, each item contains a lengthy paragraph containing facts and several questions related to the paragraph. The model needs to predict the range of characters that answer the questions. In SQuAD-v2, the problem becomes more challenging that the questions might be unanswerable by the context. We follow the same data processing script as BERT/RoBERTa for fair comparison; more details about the training process are described in the Supplementary material. The experiment results are presented in Table 5. As we can see, the FLOATER model beats the baseline RoBERTa model consistently across most datasets. The improvement is significant, considering that both models are finetuned from the same pretrained checkpoint.

4.3. More Discussions and Analysis

How inductive is FLOATER? FLOATER is designed to be inductive by a data-driven dynamical model (8). To see how inductive FLOATER is when comparing to existing approaches, we design the following experiment. We first notice that in WMT14 En-De dataset, 98.6% of the training sentences are shorter than 80 tokens. Based on that, we make a new dataset called *En-De short to long* (or S2L for brevity): this dataset takes all the short sentences (< 80 tokens) as the training split and all the long sentences (≥ 80 tokens) as the testing split. We further divide the testing split to four bins according to the source length fallen in $[80, 100)$, $[100, 120)$, $[120, 140)$, $[140, +\infty)$. BLEU scores are calculated in each bin, and the results are presented in Figure 2.

Our FLOATER model performs particularly well on long sentences, even though only short sentences are seen by the model during training. This empirical observation supports our conjecture that FLOATER model is inductive: the dynamics learned from shorter sequences can be appropriately generalized to longer sequences.

Is RNN a good alternative to model the dynamics? Recurrent neural network (RNN) is commonly used to perform sequential modeling. RNN and our continuous dynamical model (8) indeed share some commonality. Computing the value at the i-th step relies on the results at the $(i-1)$-st step. Further, they all contain trainable parameters, allowing them to adapt to each particular task. Lastly, they can be extrapolated to any length as needed. To see if RNN works equally well, we model the sequence $\{\boldsymbol{p}_i\}_{i\in\{1,2,\dots\}}$ with RNN models:

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

where $\boldsymbol{z}_i \in \mathbb{R}^{d_{\text{in}}}$ is the input to the RNN model at index i. Recall in RNN language models, $\boldsymbol{z}_i$ is the word embedding or hidden feature of the i-th token. In our case, since we apply RNN to learn the encodings as opposed to hidden features, sensible inputs can be scalar value i or vectorized value $\mathrm{Vectorize}(i)$ by sinusoidal encoding. We tried both choices on WMT14 En-De data and found that vectorized value generally works better, though not as good as our FLOATER model. Detailed results can be found in Table 6.

What does each position encoding look like? To better understand how different position encodings affect the sequence modeling, in Figure 3, we visualize the position embedding matrix $\boldsymbol{p}$ obtained from four different position encoding approaches for the Transformer-base backbone on WMT14 En-De dataset. We can see that sinusoidal encoding (3a) is the most structural, while position embedding (3b) is quite chaotic. Our FLOATER model learns position representation completely from data, but still exhibits some regularities (3c). Finally, the RNN model (3d) fails to extract sufficient positional information, probably due to the vanishing gradient problem. Another finding is that by looking at (3b), we observe that the vectors are nearly constant among different large positions (near the bottom of Figure 3b, we see patterns of vertical lines with the same color).

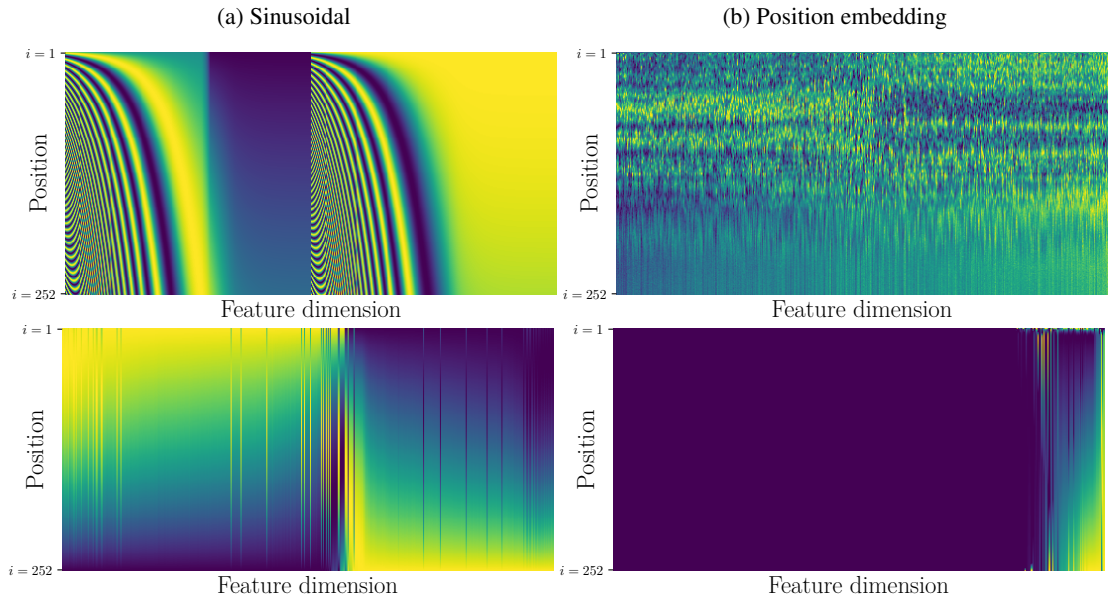

Figure 3. Visualizing the four different position methods. All models are trained using the Transformer-base architecture and En-De dataset. For better visualization, dimension indices are permuted in Figure 3b-3d.

This phenomenon is due to long sentences in the dataset being scarce, and so the positional information carried by lower indices cannot be extrapolated to higher indices. On the contrary, the dynamical model proposed in this paper enjoys the best of both worlds – it is adaptive to dataset distribution, and it is inductive to handle sequences with lengths longer than the training split.

4.4. Remarks on Training and Testing Efficiency

It is not surprising that during the training time, our flow-based method adds a non-negligible time and memory overhead; this is because solving the Neural ODE precisely involves $\sim$100 times forward and backward propagations of the *flow model*. Even though we deliberately designed a small flow model (consisting of only two FFN and one nonlinearity layers), stacking them together still increases training time substantially. To make it possible to train big models, we use the following optimizations:

- Initialize with pretrained models that do not contain flow-based dynamics, as discussed in Section 3.3.
- From (8), we know that if $\boldsymbol{h}(\cdot)$ is close to zero, then the position information diminishes (derived in appendix). In this way, our model degenerates to the original Transformer. Inspired by this property, we can initialize the FLOATER with smaller weights. Combining with the previous trick, we obtain an informed initialization that incurs lower training loss at the beginning.
- We observed that weights in $\boldsymbol{h}(\cdot)$ are more stable and easy to train. Thus, we can separate the weights of $\boldsymbol{h}(\cdot)$ from the remaining parts of the Transformer model. Concretely, we can 1) cache the positional bias vectors for some iterations without re-computing, 2) update the weights of flow models less frequently than other parts of the Transformer, and 3) update the flow models with a larger learning rate to accelerate convergence.
- For the RoBERTa model, we adopt an even more straightforward strategy: we first download a pretrained RoBERTa model, plug in some flow-based encoding layers, and re-train the encoding layers on WikiText-103 dataset for one epoch. When finetuning on GLUE datasets, we can choose to freeze the encoding layers.

Combining those tricks, we successfully train our proposed models with only 20-30% overhead compared to traditional models, and virtually no overhead when finetuning RoBERTa model on GLUE benchmarks. Moreover, there is no overhead during the inference stage if we store the pre-calculated positional bias vectors in the checkpoints.

5. Conclusions

In this paper, we have shown that learning position encoding with a dynamical model can be an advantageous approach to improve Transformer models. Our proposed position encoding approach is inductive, data-driven, and parameter efficient. We have also demonstrated the superiority of our proposed model over existing position encoding approaches on various natural language processing tasks such as neural machine translation, language understanding, and question answering tasks.

Acknowledgement

This work is partially supported by NSF IIS-1719097. Xuanqing Liu would like to thank Qie Hu for helpful discussions.

References

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Rajpurkar, P., Jia, R., and Liang, P. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Shaw, P., Uszkoreit, J., and Vaswani, A. 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)*, pp. 464–468, 2018.

Tenenbaum, M. and Pollard, H. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. Dover Books on Mathematics. Dover Publications, 1985. ISBN 9780486649405. URL https://books.google.com/books?id=iU4zDAAAQBAJ.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, B., Zhao, D., Lioma, C., Li, Q., Zhang, P., and Simonsen, J. G. Encoding word order in complex embeddings. *arXiv preprint arXiv:1912.12333*, 2019.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.