Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding: Appendix

Anonymous Author(s)
Affiliation
Address
email

1 Attention-Based Models

We review positional encoding in the context of Transformer models [5]. The central building block of these models is multi-head attention and each attention head is calculated as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$ (1)

where queries $Q \in R^{N \times d_k}$, keys $K \in R^{N \times d_k}$, and values $V \in R^{N \times D_v}$. $N$ is the number of items to consider, e.g., the number of tokens in a sequence or the number of pixel patches in an image. $d_k$ is the dimension of a key and query, and $D_v$ is the dimension of a value vector. Queries, keys and values are acquired via a linear projection of the input at each attention layer. For self-attention, they share the same input:

$$Q = E_X M_Q; K = E_X M_K; V = E_X M_V$$ (2)

where $M_Q \in R^{|E_X| \times d_k}$, $M_K \in R^{|E_X| \times d_k}$ and $M_V \in R^{|E_X| \times d_v}$ are the linear projection. $E_X \in R^{N \times |E_X|}$ is the embedding of input $X$, which is jointly represented by its content embedding, $C_X$, and its positional encoding, $P_X$.

$$E_X = C_X \oplus P_X$$ (3)

where $\oplus$ can be either concatenation or element-wise addition. Previous work has investigated different combinations and decomposition of positional encoding and content embedding [2]. While concatenation and addition provide comparable results, the lack of positional encoding, $P_X$, will cause a significant drop in accuracy [5][1]. In this paper, we investigate methods for realizing $P_X$.

2 Hyperparameters & Parameter Sizes

For Reformer experiments, each model is based on the Reformer model for the Imagenet64 task [3]. The number of parameters for each Reformer model is summarized in Table 1. We here focus on the positional encoding part of the model that is where each approach differs. Our positional encoding uses roughly the same number of trainable parameters as Embed-2D, the benchmark method used in the original Reformer. For all the Fourier-based methods, including Learnable-Fourier+MLP, we used $|F| = 768$, $|H| = 32$, $D = 768$ and $\gamma = 1.0$. For the MLP modulators, we used LayerNorm before the dense projections, $W_1$ and $W_2$ (Algorithm 1 Line 2), and a dropout rate of 10% after ReLu, the non-linear activation. We set $G = 1$ because vertical and horizontal positions need to be mapped jointly to model the inductive bias of L2 distances on an image. Embed-1D uses significantly more parameters because it needs to assign an embedding vector for each position in a flattened image. Sine-1D and Sine-2D are parameter-free encoding, thus use the least parameters. We follow

the experimental procedure as detailed in the Reformer paper. All our experiments used a 6-layer, 8-head-attention Reformer, with $d_{\text{model}} = 1024$, $d_{ff} = 4096$, and $n_{\text{heads}} = 8$. These models are implemented based on the public Reformer codebase in Trax\footnote{https://github.com/google/trax/tree/master/trax/models/reformer}. The training for each Reformer model is parallelized across 32 TPU v2 cores, and each batch contains 8 sequences (images) on each core. We trained each model variant for 100k steps, which took about 24 hours to complete.

The parameter sizes for each DETR model\footnote{https://github.com/facebookresearch/detr/blob/master/models} are shown in Table\footnote{https://github.com/google/jax}. All the variants of DETR roughly uses the same number of trainable parameters. We used $\gamma = 1.0$ for Learnable-Fourier + MLP. The MLP uses a dense layer $2 \times 256$ with ReLu as activation, which is then followed by another dense layer of $256 \times 256$. We did not use any dropout in these methods. We use the default 6-layer Encoder-Decoder setup in DETR, using all the same hyperparameters, which uses COCO 2017 detection dataset with 118k images for training and 5k for validation. All the variants are based on the DETR codebase\footnote{https://github.com/google/jax} which are ported into JAX\footnote{https://github.com/google/jax}, a library for machine learning. The training for each DETR model is parallelized across 64 TPU v3 cores with a batch size of 64 images. We follow the experimental procedure of the DETR paper. We trained each variant for 90k steps, which took roughly 16 hours to complete.

For UI Widget Captioning experiments, the number of parameters of each model variant is shown in Table\footnote{https://github.com/google/jax}. The model architecture that is shared by each model variant is summarized in the paper.
and detailed in the previous paper [4]. For Fourier-based methods, we used $|F| = 128, 64, 32$, $G = 1, 2, 4$ for position grouping variants: $1/4, 2/2$ and $4/1$, respectively. We used $\gamma = 100$ for initializing $W_r$ for all the Fourier-based methods. We used a dropout of 20% after the non-linear activation in the MLP modulator. We use the same model architecture and hyperparameters of the strongest model, Pixel+Local+Context, as the original paper [4], and built our experiment based on the public codebase of widget captioning. Specifically, the screen encoder uses a 6-layer, 8-head Transformer with a hidden size of 128. We train all the models to 100k steps with Adam optimizer and a scheduled learning rate detailed the original paper. All the models converged within 12 hours using 4 NVIDIA V100 GPU.

3 Analyzing Learned Positional Encoding

Our positional encoding is seeded with Fourier features, which can evolve as learning progresses. In this section, we analyze the positional encodings learned from the Reformer and Widget Captioning tasks.

3.1 PE Analysis for Imagenet64 Reformer Tasks

Figure 1 visualizes the similarity of a given position on a $64 \times 64$ image, to the rest of the positions on the image, at the initial stage and the end of the training. The similarity is computed based on the dot product of the positional encoding of these positions. The first row, Init, shows the similarity heatmap resulted from the initially seeded Fourier features based on $\gamma = 1.0$. The second row,
3.2 PE Analysis for Widget Captioning Tasks

Positional relationships are more complex in the Widget Captioning task, because each position is defined as a four-coordinate bounding box. We consider point-wise similarity a building block for bounding box similarity as discussed in the paper. Figure 3 shows the point-wise positional similarity learned by Learned-Fourier+MLP 2/2, which groups four coordinates into two groups to represent the top-left corner and the right-bottom corner of a bounding box. In this task, we see a more spread positional relationship than that of the Imagenet64 Reformer task, because we seed the Fourier features with \( \gamma = 100 \). We observed that the positional relation becomes more concentrated over the course of the training than that of the initial encodings. We also see the positional relation distribution becomes more skewed (towards the anti-diagonal direction). To understand whether...
maintain the symmetry of the distribution would help on accuracy, we conduct additional experiments by applying a regularizer to the Fourier weights $W_r$ (Algorithm 1 Line 1) as the follow.

$$L_{KL} = -\frac{1}{2}(1 - \log \bar{\sigma}^2 + \log \sigma^2 - \frac{\sigma^2 + \mu^2}{\bar{\sigma}^2})$$ \quad (4)

where $\mu$ and $\sigma^2$ are the mean and variance of $W_r$. $\bar{\sigma}^2$ is the target variance that is also learnable, which is initialized as $\gamma^{-2}$. The KL loss ensures $W_r$ to obey a Gaussian distribution centered at 0 thus maintains the symmetry of positional relationships along all the directions. When training the model, the regularizer loss $L_{KL}$ is added to the overall loss for optimization.

$$L_{total} = L_{model} + \alpha L_{KL}$$ \quad (5)

In this experiment, we use $\alpha = 1$. The resulted positional encoding is shown in Figure 4. As we can see, the symmetry of the positional relation distribution is better maintained with the KL loss. We see a clear improvement of accuracy with the use of this KL loss for Learned-Fourier+MLP 2/2. However, using the KL loss does not seem to help image-based tasks much (Imagenet64 and DETR tasks). We suspect that as shown in Figure 1, the symmetry of positional relation distribution is naturally maintained even without using the KL loss. Thus KL loss is less useful in such cases.

**References**


