Appunti del paper *BP-Transformer: Modelling Long-Range Context via Binary Partitioning*

A.A. 2023/2024

Rosso Carlo

Contents

1 Introduction

The Transformer model is widely successful on many natural language processing tasks. However, the quadratic complexity of self-attention limit its application on long text. In this paper, adopting a fine-to-coarse attention mechanism on multi-scale spans via binary partitioning (BP), we propose BP-Transformer. BPT yields to $O(K \cdot log(n/k))$ complexity, where n is the sequence length and k is a hyperparameter to control the density of attention. Note that if $k = n$, BPT degenerates to the vanilla Transformer.

The dependency relations between token are totally learned from scratch. Therefore, Transformer usually performs better on huge datasets and is easy to overfit on small datasets (there is a citation, for this statement). The above observation motivates us to explore better structure for self-attention models to balance the capability and computation complexity. We propose a new architecture, BP-Transformer, which partitions the input sequence into different multi-scale spans via binary partitioning (BP). BPT incorporates an inductive bias of attending the context information from fine-grain to coarse-grain as the relative distance increases.
Thus, BPT has the following advantages over the vanilla Transformer:

- long-range context in an hierarchical fashion;
- reduces computation cost with fewer edges;
- introduces coarse-to-fine connections to approximate the reasonable inductive bias of the language.

We show that the inductive bias of BPT works nicely on short text and can scale to large datasets.

2 Related Work

2.1 Transformer

The Transformer model revolutionized NLP by leveraging self-attention mechanisms to process sequences of data. Here's a breakdown of its core components:

- **Input Representation**: It is used an embedding layer that converts words or subwords into dense vectors;
- **Positional Encoding**: Since the Transformer doens't inherently capture the order of tokens, positional encodings are added to the input embeddings to provide information about the position of each token in the sequence;
- **Multi-head Self-Attention**:
  - **Self-Attention Mechanism**: Self-attention allows each token to attend to every other token in the sequence, enabling the model to capture dependencies regardless of distance. This is done by computing attention scores between pairs of tokens and then using these scores to weight the importance of each token's contribution to the representation of the others;

  – **Multi-head Mechanism**: Instead of using a single attention mechanism, the Transformer uses multiple attention mechanisms (or heads) in parallel.

  – **Formulas**: Given a sentence with n input tokens, the Transformer model iteratively computes at layer t the d-dimensional representations of each input token $H^t \in \mathbb{R}^{n \times d}$, where H^0 represents the initial token embeddings. The core of a Transformer step is Multi-head Self-Attention (MSA), which can be formulated as follows:

$$\begin{aligned} \mathsf{MSA}(H) &= [\mathsf{head}_1, \ldots, \mathsf{head}_h]\, W^O, \\ \mathsf{head}_i &= \mathsf{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i, \\ Q_i = HW_i^Q, \quad K_i &= HW_i^K, \quad V_i = HW_i^V \end{aligned}$$

    Where h is the number of heads, and W_i^Q, W_i^K, W_i^V, W^O are learnable parameters;

- **Feed-Forward Networks**: Transformer then computes H^{t+1} from H^t as follows:

$$Z^t = \mathsf{norm}\left(H^t + \mathsf{MSA}(H^t)\right), \tag{2.1}$$

$$H^{t+1} = \mathsf{norm}\left(Z^t + \mathsf{FFN}(Z^t)\right) \tag{2.2}$$

  where norm$\cdot$ is layer normalization and FFN stands for Position-wise Feed-Forward Networks. Note that each step t has its own parameters;

- **Layer Normalization and Residual Connections**: Each sub-layer (self-attention and feed-forward) if followed by layer normalization and residual connections to stabilize training and help the model learn more effectively;

- **Stacking Layers**: The Transformer model stacks multiple identical layers of self-attention and feed-forward networks. This allows the model to learn increasingly abstract representations of the input.

2.2 Lightweight Self-Attention

Recently there has also been several works focusing on reducing the computational cost of Self-Attention mechanisms. Sparse Transformer decomposes attention into two categories: for a sequence of length n, we divide it into $\sqrt{n}$ equal-sized blocks. Each token attends to its previous tokens inside a $\sqrt{n}$ block it lies in, and to $\sqrt{n}$ previous blocks.

Transformer-XL introduces the notion of recurrence into Transformer. It divides the input sequence into multiple segments and recurrently attends to the hidden states of the previous segments. Transformer-XL can only model sequences in one direction, making it hard to deal with tasks where bi-directional information is required.

3 Proposed Model

In this paper, we balance the model capability and computation complexity by incorporating the inductive bias. The key insight is that instead attending to every token, the input token attends to different spans away from it in a fine-to-coarse fashion.

3.1 Transformer as Graph Neural Networks

A valid perspective is to view information fusing with self-attention in Transformer as message passing on a fully-connected graph, with input tokens as nodes and attentions between nodes as edges (there is a citation). Thus, different graph structure encodes different inductive bias of attention and results in different time/space complexity.

To describe Transformer in GNN framework, we first construct a fully-connected graph $\mathcal{G}$, in which each node is a token of the input sequence. All nodes in $\mathcal{G}$ are interconnected and each node has a self-loop edge.

We extend the self-attention mechanism of Transformer to graph, called Graph Self-Attention (GSA). For a given node u we update its representation according to its neighbour nodes, formulated as $h^u \leftarrow \text{GSA}(\mathcal{G}, h^u)$. Let $\mathcal{A}(u)$ be the set of the neighbour nodes of u in $\mathcal{G}$, $\text{GSA}(\mathcal{G}, h^u)$ is detailed as follows:

$$A^u = concat\left(\{h_v \mid v \in \mathcal{A}(u)\}\right), \tag{3.1}$$

$$Q_i^u = H_k W_i^Q, \quad K_i^u = A^u W_i^K, \quad V_i^u = A^u W_i^V, \tag{3.2}$$

$$\text{head}_i^u = \text{softmax}\left(\frac{Q_i^u K_i^{uT}}{\sqrt{d_k}}\right) V_i^u, \tag{3.3}$$

$$\text{GSA}(\mathcal{G}, h^u) = [\text{head}_1^u, \ldots, \text{head}_h^u] W^O, \tag{3.4}$$

where d is the dimension of h, and W_i^Q, W_i^K, W_i^V, W^O are learnable parameters of the i-th head.

3.2 Node Construction

To achieve the effect of fine-to-coarse attention, we partition a sequence into mulit-granular spans via binary partitioning (BP). Each partition can be regarded as a node in GNN and its representation is computed according to its contained tokens. Each leaf corresponds to an input token in the sequence.

We simply divide the nodes into two types: token and span.

3.3 Edge Construction

Formally, let $u_{l,m}$ denote the $m-th$ node at level l. The level of token nodes is set to 0. A span node $u_{l,m}$ represents a partition consisting of token nodes $u_{0,2^l \cdot m+1}, \ldots, u_{0,2^l \cdot (m+1)}$.

We construct two kind of edges:

Affiliated Edges For each span node $u_{l,m}$, we add a directed edge from each of its contained token nodes ($u_{0,2^l \cdot m+1}, \ldots, u_{0,2^l \cdot (m+1)}$) to $u_{l,m}$. The role of afiiliated edges is to shorten the path between a span node and its corresponding token nodes.

Contextual Edges For a leaf node $u_{0,i}$, we add the incoming edges from the different granularity. For simplicity, we describe the process of constructing edges from its right context of node $u_{0,i}$. The left context is similar. We use a hyperparameter k to determine the connection density of the graph. We add k edges per level to capture the information from the right context. For node $u_{0,i}$ its contextual nodes are

$$\begin{aligned}
&u_{0,p_0}, \dots, u_{0,p_0+k-1} \\
&u_{1,p_1}, \dots, u_{1,p_1+k-1} \\
&\cdots \\
&u_{l,p_l}, \dots, u_{l,p_l+k-1} \\
&\cdots
\end{aligned}$$

where p_l is the start index at level l and can be computed recursively: $p_l = \text{parent}(p_{l-1} + k)$ and $p_0 = i + 1$. We can see that the distances between any two token nodes are no greater than 2 in graph $\mathcal{G}$.

3.4 Graph Update

The representations of span nodes are initialized with all zeros, while the representations of token nodes are initialized with the corresponding word embeddings. And it is used the same GSA mechanism to update the representations of all nodes in $\mathcal{G}$.

3.5 Relative Positional Encoding

We modify Equation 3.3 to include positional representations:

$$\begin{aligned}
R^u &= \text{concat}\left(\{r_v \mid v \in \mathcal{A}(u)\}\right), \\
head_i^u &= \text{softmax}\left(\frac{Q_i^u (K_i^u + R^u)^T}{\sqrt{d_k}}\right) V_i^u.
\end{aligned}$$