Landmark Attention

Random-Access Infinite Context Length for Transformers

Amirkeivan Mohtashami, Martin Jaggi

Machine Learning and Optimization Laboratory

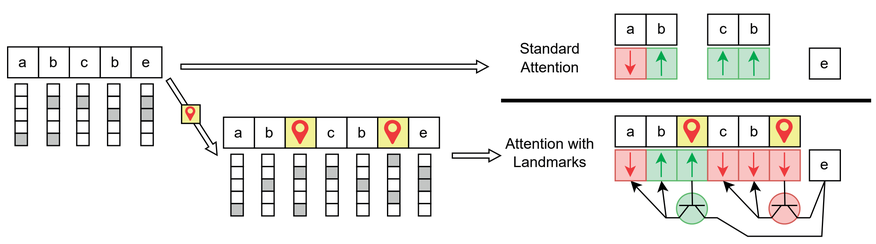
Overview

Problem

- Attention offers the flexibility of accessing the represetation of any other token in each layer.
- This flexibility comes with quadratic computational cost and highly problematic memory footprint, limiting the context length.
- Previous approaches to remove this limit often sacrifice the random-access felxibility of attention.

Solution

- We break the input into blocks of fixed length and introduce a special token for each block, called a landmark, which acts as a gate for attending to its corresponding block.
- The gating mechanism is controlled by the attention score to the landmark token.
- At inference time, we compute the attention scores of the landmarks and retrieve the blocks corresponding to the highest scoring landmarks (active gates), integrating them into the attention.
- Our proposed approach maintains the random-access flexibility of attention and offers an alternative solution to the recurrent memory approaches.



Training

- We add a landmark token after every l_{block} tokens. The initial embedding for the landmark token is trained (similar to any other token), thus the vocabulary size is increased by one.
- Each landmark token acts as a gate for attending to the preceding l_{block} tokens.
- Landmark tokens are passed through the transformer as any other token while their representations are updated using the self-attention mechanism.
- To have the gating mechanism, we want the attention weight for a token to depend on the similarity of the query vector with both the token's key and with the key of its block's landmark token.
- First we replace the softmax function in the attention layer with **Grouped Softmax** which computes the softmax separately over each group.

$$\sigma_G(\mathbf{v}, \mathbf{g})_i := \text{GroupedSoftmax}(\mathbf{v}, \mathbf{g})_i := \frac{e^{\mathbf{v}_i}}{\sum_{j:\mathbf{g}_j=\mathbf{g}_i} e^{\mathbf{v}_j}} .$$

Grouping Scheme

- For each block, we create a sepearte group and put its regular (non-landmark) tokens in that group.
- When computing the attention weights for the i-th token, landmark tokens for other blocks are placed in the same group as the i-th token.
- The landmark token for the i-th token's block is **ignored** when computing the attention weights for the i-th token.

$p_i \longrightarrow$ index of landmark token corresponding to i-th token's block

$$\mathbf{G}_{i,j} := \begin{cases} p_j & p_j \neq j \\ -1 & p_i = j \\ p_i & \text{otherwise} \end{cases}$$

Landmark Attention

- Using the above grouping, the attention weight for each token can be computed as the product of the token's attention weight and it's corresponding landmark token's attention weight.
- Under this scheme, the attention weights sum to one same as in the standard softmax function.
- Since tokens in the same block and the landmark tokens share the softmax group, the model has to choose between attending to other blocks and current tokens.
- Intuititvely, the grouping forces the model to only attend to relevant blocks because of this trade-off.

$$\begin{aligned} \mathbf{S}_{i,j} &:= \text{SoftmaxScore}(\mathbf{Q}, \mathbf{K})_i \\ &:= \text{GroupedSoftmax}\left(\frac{\mathbf{Q}_i^\top \times \mathbf{K}}{\sqrt{d_{\text{head}}}}, \mathbf{G}_i\right) \end{aligned}$$

$$\text{Att}(\mathbf{Q}, \mathbf{K})_{i,j} := \begin{cases} 0 & p_j = j \\ \mathbf{S}_{i,j} & \mathbf{G}_{i,j} = \mathbf{G}_{i,i} \wedge p_j \neq j \\ \mathbf{S}_{i,j} \cdot \mathbf{S}_{i,p_j} & \mathbf{G}_{i,j} \neq \mathbf{G}_{i,i} \wedge p_j \neq j \end{cases}$$

i	0	1	2	3	4	5	6	7	8
mask + $\frac{Q_6^T K}{\sqrt{d_{\text{head}}}}$	1	1	1	1	1	1	1	mask	mask
p_i	2	2	2	5	5	5	8	8	8
$G_{6,j}$	2	2	8	3	3	8	8	8	-1
$S_{6,j}$	0.5	0.5	0.33	0.5	0.5	0.33	0.33	0	ign
$W_{6,j}$	0.167	0.167	0	0.167	0.167	0	0.33	0	0

Results

Pass Key Retrieval Task

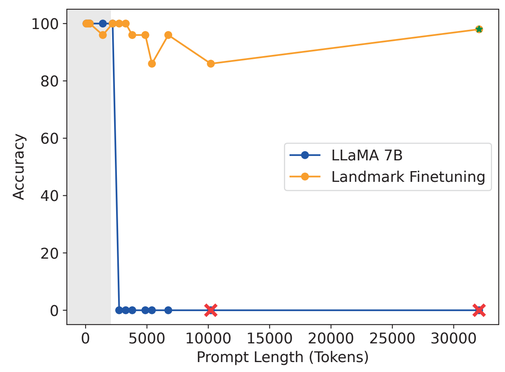
- The task is to find a pass phrase hidden in a long piece of text.
- Instances of this task are generated by randomly choosing the pass key (a number between 1 and 50000) as well as its location.
- Prefix and suffix strings filled with a repetitive text are added to the pass key to ensure the desired length.

```
There is an important info hidden inside
a lot of irrelevant text.  Find it and
memorize them.  I will quiz you about the
important information there.
<prefix filler by continuously repeating:
The grass is green.  The sky is blue.  The
sun is yellow.  Here we go.  There and back
again.>
The pass key is <PASS KEY>.  Remember it.
<PASS KEY> is the pass key.
<suffix filler>
What is the pass key?  The pass key is
```

- We use the success rate as the evaluation metric.

Evaluation

- We fine-tune LLaMA 7B using our method.
- The original model fails to retrieve the pass key and even runs out of memory as the prompt gets longer (marked with a red cross)
- In contrast, the fine-tuned model using landmarks can successfully identify the pass key with a high success rate.
- A more efficient inference mechanism allows offloading KV-Cache to CPU which resolves the memory problem further (a green star marks this mechanism's deployment).



- The results demonstrate that our method can be successfully used even during fine-tuning to extend the context length limit to arbitrary large values.

Inference

- Similar to training, the input gets augmented by a landmark token after every l_{block} tokens.
- Then the input is broken into chunks of l_{local} length. The chunks are iteratively fed to the model from the beginning to the end.
- To retrieve relevant blocks, each attention layer has access to a cache of previous blocks. The cache stores the key-value vectors for all tokens of those blocks, including the landmark tokens.
- When processing each chunk at each layer, we first compute the attention score of each token with the landmark tokens currently in the cache.
- For each token, we compute the attention score for all the tokens in the top-k highest scoring blocks and prepend the obtained attention matrix to the local attention matrix.
- We finish by applying GroupedSoftmax with the same grouping as in training to the attention matrix and computing the weighted average of the value vectors to obtain the token's representation.

Stingy Position Mapping

- Optimally, the tokens are encoded with their actual position index. However, a known flaw of Transformers is their inability to extrapolate to lengths not observed during training.
- Previous methods proposed to alleviate this flaw usually penalize or prevent attention to long distance tokens.
- We instead use a special approximate position mapping scheme since our main goal is to facilitate attending to large contexts. We allocate a segment with length (k+1) · l_{block} in the beginning of the context. We index the current chunk starting after this segment.
- We map the index for the latest k blocks to the corresponding place within the last k blocks in the allocated prefix segment. All other blocks in the cache have their position mapped to the first block in the allocated prefix segment.
- Once we decide which blocks to retrieve, we map those among the latest k blocks to the right of pre-allocated segment and map the rest of the blocks to the left of the pre-allocated segment, while respecting the order of the blocks.