Sequoia: Hierarchical Self-Attention Layer with Sparse Updates for Point Clouds and Long Sequences

Hugo Sonnery * [1], Thuan Anh Trang * [2], Thieu N. Vo [3], Siamak Ravanbakhsh [1], and Truong Son Hy [4]

McGill University / Mila [1], FPT Software AI Center [2], Ton Duc Thang University [3], University of California San Diego [4]

Co-first author *

Abstract

Lack of efficiency of self-attention in dealing with **long inputs** has motivated many **efficient transformers**. We consider *hierarchy as an interesting route toward sparsity.*

This can be achieved by forming a tree on top of the input tokens and limiting the span of attention, based on the branching factor, thereby reducing the complexity of the self-attention layer to $O(n\log(n))$. Our proposed method called `Sequoia` leverages the hierarchy within a single self-attention layer and **sequentially applies the attention mechanism to the tree in a bottom-up fashion**. Our preliminary results on point cloud tasks and sequence classification benchmarks suggest favourable performance compared to state-of-the-art transformer architectures without resorting to efficient CUDA implementation.

Background

At the basis of transformers is the mechanism of self-attention, which iteratively refines tokens' embeddings using a weighted *all-to-all communication* scheme :

$$\mathcal{A}(i,j) = \mathcal{S}(Q_i, K_j) = \frac{\exp(Q_i K_j^T)}{|\mathcal{S}_i|} \times \frac{1}{\sqrt{d_k}} \tag{1}$$

These interaction weights (also called attention scores if they are normalised) are used to update each token's embedding :

$$X_i' = \mathcal{A}_{W_Q,W_K,W_V}(X,i) = \sum_{j\in[1,N]} \mathcal{S}(Q_i,K_j)V_j \tag{2}$$

The quadratic complexity in the sequence length ($\Omega(n^2)$) of $\mathcal{A}$'s computation makes **self-attention intractable for long sequences**.

Most efficient transformers either *simplify* the computation of the attention matrix or *get rid of it altogether* :

- **Content-based sparse attention** (e.g. Reformer) matches tokens together into buckets using locality-sensitive hashing before applying local attention.
- **Sparsity heuristics** transformers get rid of the bulk of the computations in the dot-product attention in a structured way (e.g. only computing local attention scores), allowing for efficient implementation without custom CUDA kernels.
- **Learnable sparsity patterns** transformers (e.g. Sinkformer) induce sparsity by sorting the keys and values matrices such that local heuristics can be applied adaptively.

References

[1] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. An attentive survey of attention models. ACM Transactions on Intelligent Systems and Technology (TIST), 12(5):1–32, 2021.

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 652–660, 2017

[3] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient transformers. arXiv preprint arXiv:2011.04006, 2020

[4] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1912–1920, 2015.

[5] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 909–918, 2019

Hierarchical Transformers

Our proposed attention scheme consists in **sequentially applying an attention-based update** at various levels of a k-ary tree constructed on top of the input tokens.

- The **leaves** of the tree correspond to **actual tokens** of the input.
- **Intermediate nodes** are **hubs for information propagation** at different resolutions, thus allowing global information sharing

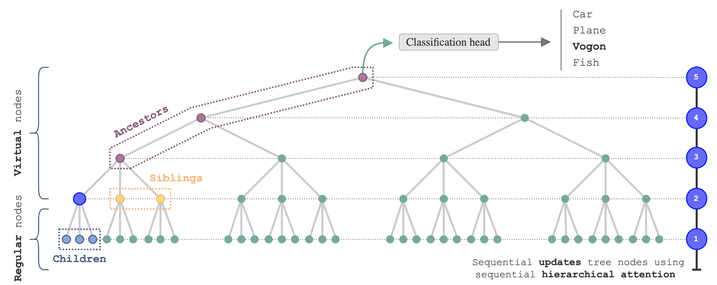

Figure 1: `Sequoia`, our proposed sequential hierarchical attention scheme.

The construction of this tree depends on the geometry of the input data itself :

- For **point cloud** data, we can use Euclidean distance for deciding the neighbourhoods (KD-tree, hierarchical clustering or hierarchical K-nearest neighbours).
- For **sequential** data, tokens' timestamps define neighbourhoods. Higher nodes in the tree will represent increasingly larger hierarchical temporal intervals.

Assuming a k-ary tree $\mathcal{T}$ has been constructed, the most relevant attention weights to compute in order to apply self-attention are the interaction scores of node i with its **children**, **siblings** and **ancestors**. Global information sharing is achieved by **sequentially updating the nodes' embeddings** in a bottom-up fashion across the tree, all in a single layer. More concretely, $\mathcal{A}^{\text{children}}(i)$ refines node i's embedding by computing scalar dot-product attention between its query Q_i and its children' keys K_j (respectively siblings and ancestors) :

$$\mathcal{A}^{\text{children}}(i) = \text{Attention}\left[Q_i, (K_j, V_j)_{j\,\in\,\text{children}(i)}\right] \tag{3}$$

The three proposals are then pooled together into a single updated embedding for node i. The complexity in both run-time and memory is quasi-linear in the input size ($\mathcal{O}(n\log(n))$).

Applications in long-range sequence classification

In order to accurately measure the quality of the inductive bias of sequential hierarchical attention, we train our model on an array of complementary classification tasks from the Long Range Arena benchmark: `IMDB` (text), `Cifar10` (image), `Listops` (mathematical sequence parsing), `Retrieval` (byte-level document matching) and `Pathfinder` (image).

Method	IMDB	Cifar10	Listops	Retrieval	Patfhinder-32
Nystrom-64	62.11	56.89	36.64	81.55	75.62
Linformer-64	56.03	43.57	38.66	76.22	90.22
Performer-64	62.46	38.59	18.4	78.62	69.90
Linear	50.98	22.28	17.79	49.41	50.36
Softmax attention	60.87	48.45	39.62	`OOM`	88.61
None attention	60.50	37.03	37.1	80.42	50.36
Sequoia (ours)	**62.72**	**49.88**	**37.70**	**67.04**	**58.44**

Figure 2: Sequence Classification results on Long Range Arena benchmark for efficient transformers.

Applications in point clouds

To demonstrate the efficiency of Sequoia on point clouds, we conduct experiments on **shape classification** and **part segmentation**. In each task, we compare the performance of our model with the normal softmax attention and other models specifically designed for point clouds. Although we do not produce the SOTA results on the two tasks for point cloud, `Sequoia` is a lightweight attention-based model with competitive results.

Method	Accuracy	mAcc
VoxNet	85.9	83.0
MVCNN	90.1	–
PointNet	89.2	86.0
DGCNN	92.9	90.2
GridGCN	93.1	91.3
Set Transformer	90.4	–
PointNet++	91.9	88.4
PointConv	92.5	–
KPConv	92.9	–
PointTransformer	93.7	90.6
Softmax attention	89.5	87.4
Sequoia	**92.0**	**88.4**

Figure 3: Shape Classification results on ModelNet40

Method	Inst IoU	Class IoU
PointNet	71.9	43.7
DGCNN	85.1	82.3
PointNet++	85.1	81.9
PointConv	85.7	82.6
PointCNN	86.1	84.6
PointTransformer	86.6	83.7
Softmax attention	78.8	74.9
Sequoia	**82.7**	**79.5**

Figure 4: Part Segmentation results on ShapeNetPart

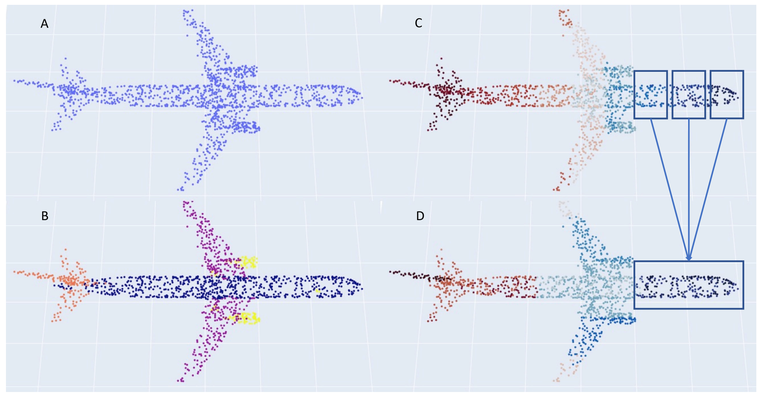

Figure 5: A. The original point cloud. B. The point cloud is segmented by the model. C. The point cloud is extracted from the lower level of the tree (The point cloud is divided into many small clusters). D. The point cloud is extracted from the higher level of the tree (The point cloud is divided into some large clusters). For example, in Figure C, the head of the airplane is divided into 3 clusters, while there is only one cluster for this head in Figure D.

Conclusion & Future work

`Sequoia` is **general-purpose** and requires little adaptation to a particular input as long as a hierarchical k-ary tree can be built over the tokens. We are actively exploring :

- Node-wise **graph classification**.
- **Autoregressive** generation for long-range language modeling.
- Generalizing the tree structure to a **lattice** (increasing the number of pathways between input tokens) to better control the tradeoff between expressivity and the long-range nature of the receptive field.
- Decrease the computational cost of our model by using **sparsity-aware** implementations of the scalar dot product.