Funnel-Transformer - Filtering out Sequential Redundancy for Efficient Language Processing

Sumith Dutt Mishra
Smishra9@kent.edu

Introduction:

Funnel-Transformer breaks the standard Transformer on a wide variety of sequence-level prediction tasks, including text classification, language understanding and reading comprehension. The main idea is to reduce the computational cost in the sentence length wide is novel and possibly has a large impact to the community. With the recent success of language pretraining, it is highly desirable to develop more efficient architectures of good scalability that can exploit the unlabeled data at a lower cost and the power of neural self-attention models has been pushed to new level, leading many advancements in machine learning and Natural Language Processing (NLP). Typically, with more FLOPs in longer pretraining and larger models, the performance increases for pretrained Transformer model. However, it is highly expensive to pretrain or even just finetune the state-of-art self-attention models, as they require more FLOPs (Floating Point Operations per second- higher FLOPs mean faster performance) and memory resources when compared to traditional NLP.

Transformer Architecture:

The Transformer architecture is a highly modularized neural network, where each Transformer layer consists of two sub-modules, namely the multi-head self-attention (S-Attn) and position-wise feed-forward network (P-FFN). Both sub-modules are wrapped by a residual connection and layer normalization. Given a length T sequence of hidden states h = [h1, . . . , hT], the computation of a single Transformer layer can be expressed as

h ← LayerNorm(h + S-Attn(Q = h, KV = h))
hi ← LayerNorm(hi + P-FFN(hi)), ∀i = 1, · · · , T
h = hidden states

Pretraining Objectives:

The most used pretraining objective is the masked language modeling (MLM) proposed by BERT [2]. For a length-T natural language sequence x sample from a large unlabeled set D, the MLM objective first constructs a corrupted sequence xˆ by randomly replacing 15% of the tokens of x with a special token [mask] and then trains a Transformer model to reconstruct the original x based on xˆ, i.e.,

$$\max_{\theta} \ \mathcal{J}_{\text{MLM}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log P_\theta(x_i \mid \hat{\mathbf{x}}_{\mathcal{I}}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log \frac{\exp\left(e(x_i)^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)}{\sum_{x'} \exp\left(e(x')^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)},$$

Where I = Positions of the masked tokens

x^I = dependents on I

e(x) = embedding of the token x

hi(x^I) = last layer hidden state at position i

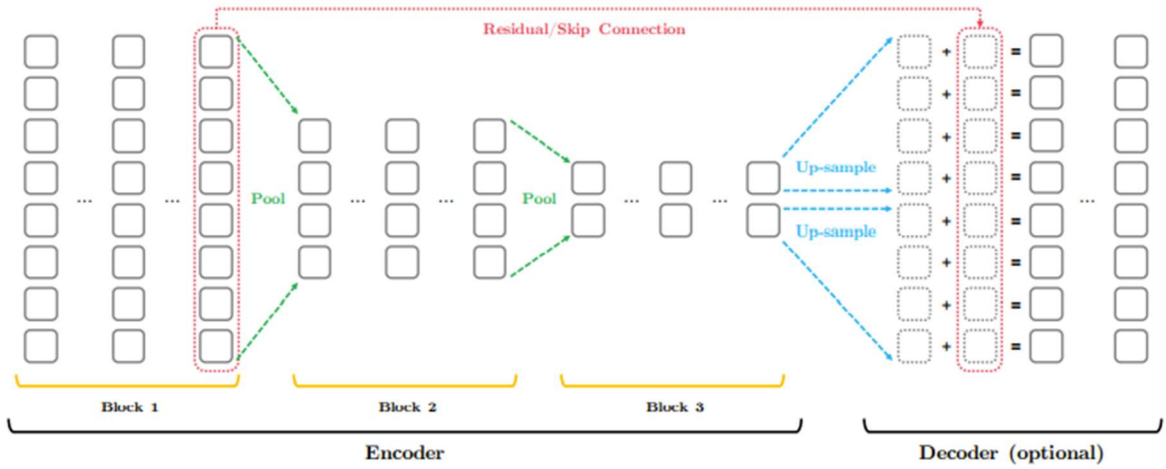
The pretraining objectives MLM by BERT should be able to produce a hidden state for each input token. Due to this requirement, we should keep full sequence of hidden states and in contrast many sequence-level downstream tasks like classification or ranking only need a single-vector summary of entire

sequence. This intern explains that we need to do compression to remove unnecessary redundance during finetuning.

Proposed Architecture:

By Funnel Transformer we design a general model which is more efficient by compressing the full sequence of hidden states into a more compact form. It can retain its ability to produce token-level representations even after compression.

This model keeps the similar architecture as Transformers enclosed self-attention and Feed Forward network sub-modules wrapped by residual connections and layer normalization to keep the high capacity and optimization advantages. To achieve the compression and computation reduction, model uses an encoder that gradually reduces the sequence length of the hidden states as the layer gets deeper. For tasks involving per-token predictions like pretraining, a simple decoder is used to reconstruct a full sequence of token-level representations from compressed encoder output.



Encoder:

- The Encoder consists of several blocks of consecutive Transformer layers.
- Within each block, the sequence length of hidden states always remains same.
- When going from low-level block to high-level block, the length of hidden sequence is reduced by performing certain type of pooling along the sequence dimension.

$$h' \leftarrow \text{pooling}(h).$$

Instead of directly feeding the pooled sequence h' into attestation layer of new block, we only use the pooled sequence to construct query vector while unpooled sequence h serves role of key and vector value.

To understand the advantage of this architecture, lets compare pool-query-only with the naïve alternative of using h' for query and key value vectors.

- Using Naive approach: The compression is controlled by pooling operation which is finished before attention module. Simple pooling methods such as average/mean pooling won't be able to achieve proper compression.
- Using Pool-query-only: The compression depends on not only how pooling is performed, but also how self-attention weighted sums the unpooled sequence to form pooled vector. Now, particular attention here can be seen as a type of linear compression that combines bases into smaller number of compression bases. So, with minimum computational overhead pool-query-only variant makes compression less cost and computation.

Let's apply simplest stride mean pooling to each sliding window of sequence. Here we will only apply with stride 2 and window size 2, the pooling operation will reduce the sequence by half and each pooled hidden state corresponds to a window of 2 unpooled hidden vectors. Once the sequence length is halved after pooling and pool-query-only attention, the rest encoder computation follows the below

$$\mathbf{h} \leftarrow \text{LayerNorm}(\mathbf{h} + \text{S-Attn}(\mathbf{Q} = \mathbf{h}, \mathbf{KV} = \mathbf{h})),$$
$$h_i \leftarrow \text{LayerNorm}(h_i + \text{P-FFN}(h_i)), \quad \forall i = 1, \cdots, T.$$

Decoder:

Decoder helps to recover the full sequence of hidden states from the encoder output of reduced length by performing up-sampling. We will apply single up-sampling (instead of multiple up-sampling with small expansion) with large expansion rate. Here, in decoder all consecutive vectors are same and do not contain token-level information. Hence, we must extract the last-layer hidden states from first block of encoder h' which has the full length and contains uncompressed token-level information.
The lower-level representation h' and up-sampled higher-level representation are added together to form a deep token-level representation g.
We use decoder only if task requires token-level prediction such as in standard pretraining or sequence labelling. If we have only vector representation of the sequence like classification, the decoder is discarded after pretraining and only encoder is finetuned.

Capacity Analysis:

With the given architecture, we can now analyze how to compress the sequence which affects the complexity and capacity of proposed model, when compared to standard Transformer.
Let's consider hidden size D with S-Attn and P-FFN, the complexity of processing a length-T sequence O(T2D + TD2). Whenever the sequence length is reduced by half in the encoder, complexity also drops. The capacity of a compressed-length layer is clearly upper-bounded by that of a normal full-length layer. If loss is generated by compression by reducing the sequence length will lead to capacity drop. The capacity drop of a single layer could be well compensated by re-investing the saved FLOPs in stacking more cheaper layers of reduced length or increasing the width of a model.

Results:

We analytically evaluate the F-TFM by first pretraining it and then finetuning it in downstream tasks. Following previous work for pretraining, we consider two common settings:

• Base scale: Pretraining models for 1M steps with batch size 256 on Wikipedia + Book Corpus. This is the setting used by original BERT. We will rely on this setting to perform fair comparison between F-TFM and the standard Transformer.
• Large scale: Pretraining models for 500K steps with batch size 8K on the five datasets used by XLNet and ELECTRA (Wikipedia + Book Corpus + ClueWeb + Gigaword + Common Crawl). We will compare F-TFM trained at this scale with previous state-of-the-art methods.
Also, Let's consider the three commonly used model sizes for standard transformer, large (L24H1024), base (L12H768) and small (L6H768) with F-TFM to evaluate the performance.

Model size	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE-AVG
L24H1024	63.2	94.8	91.8/88.5	91.1	88.7/91.7	88.7	94.0	80.5	86.6
B10-10-10	64.8	95.0	92.5/89.5	90.7	88.6/91.5	88.9	94.0	81.5	**87.0**
B8-8-8	63.5	94.7	92.2/89.0	90.7	88.9/91.7	88.8	93.6	81.2	86.7
L12H768	60.5	93.0	92.2/89.0	89.4	88.1/91.2	86.0	92.2	73.6	84.4
B6-6-6	62.5	94.0	92.2/89.0	89.5	88.4/91.4	87.0	92.7	76.5	**85.3**
B6-3x2-3x2	60.5	93.6	92.4/89.2	89.4	88.2/91.3	86.4	92.5	75.0	84.7
B4-4-4	59.1	92.7	91.8/88.7	89.1	88.2/91.3	85.5	92.0	73.2	83.9
L6H768	55.2	91.5	91.1/87.8	88.1	87.2/90.6	82.7	90.0	64.6	81.3
B3-4-4	59.0	92.8	91.8/88.5	88.5	87.8/90.9	84.8	91.8	73.2	**83.7**

Model size	IMDB	AG	DBpedia	Yelp2	Yelp5	Amazon2	Amazon5	FLOPs	#Params
L24H1024	4.440	**4.987**	0.646	1.758	28.73	2.409	32.78	1.00x	1.00x
B10-10-10	**4.404**	5.026	**0.617**	**1.734**	**28.52**	**2.400**	**32.65**	0.73x	1.22x
B8-8-8	4.552	5.079	0.664	1.713	28.84	2.438	32.87	0.58x	1.00x
L12H768	5.328	5.184	0.663	2.013	29.35	2.571	33.14	1.00x	1.00x
B6-6-6	**4.908**	**5.079**	0.654	**1.939**	**29.03**	**2.518**	**32.91**	0.88x	1.39x
B6-3x2-3x2	5.144	5.342	**0.649**	1.892	**29.03**	2.570	33.01	0.88x	1.00x
B4-4-4	5.348	5.250	0.670	1.979	29.37	2.596	33.16	0.58x	1.00x
L6H768	6.252	5.421	0.697	2.203	30.33	2.801	33.69	1.00x	1.00x
B3-4-4	**5.520**	**5.342**	**0.670**	**2.042**	**29.51**	**2.603**	**33.16**	1.00x	1.53x

Observations:

• By Giving similar or less FLOPs, by dealing sequential resolution for more layers, the F-TFM outperforms the standard Transformer in most tasks except STS-B, especially for smaller models.
• When we only compress the sequence length without increasing the depth (and #Params), F-TFM could suffer from some performance loss in certain settings on the GLUE datasets. However, as the model size increases, such performance gaps become smaller or even disappear.
• In addition, we find partial parameter-sharing often harms the performance. Therefore, the practical trade-off should be made according to the actual task and computation device.

Running time and memory consumption comparison between F-TFMs and the standard transformer on the GPU.

Sequence length	128			256			512		
Metrics	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 8 GPUs	Mem	GLUE
Batch size / GPU	64			32			16		
L12H768	1.00x	1.00x	9.2G	1.00x	1.00x	11.0G	1.00x	14.3G	84.40
B6-6-6	0.97x	0.99x	9.1G	0.95x	0.97x	10.3G	0.94x	12.5G	85.37
B6-3x2-3x2	0.93x	0.93x	8.4G	0.91x	0.92x	9.5G	0.90x	11.8G	84.78
B4-4-4	0.67x	0.67x	6.6G	0.65x	0.66x	7.5G	0.64x	9.0G	83.99
Batch size / GPU	32			12			4		
L24H1024	1.00x	1.00x	14.8G	1.00x	1.00x	14.4G	1.00x	13.9G	86.62
B10-10-10	0.87x	0.92x	14.0G	0.90x	0.93x	13.0G	0.96x	12.7G	87.03
B8-8-8	0.70x	0.73x	11.6G	0.73x	0.75x	10.8G	0.78x	10.5G	86.70

F-TFM model B6-6-6H768 achieves better results when compared to the base transformer model. Same goes with the model B6-3x2-3x2 with the same amount of base model.
B4-4-4 which has same model parameters able to provide 30%-50% speedup without losing much performance.

Conclusion:

Under the pretraining-finetuning model, we investigate a largely overlooked dimension of complexity in language processing. With the proposed Funnel-Transformer, we explained how sequential resolution can be compressed in a simple form to save computation and how the saved FLOPs can be re-invested in improving the model capacity and hence the performance.
The research is happening to improve the compression scheme, to optimize the block layout design and to re-invest the saved FLOPs which helps to improve the model more accurately.

Reference Links:

*2006.03236.pdf (arxiv.org)
Review for NeurIPS paper: Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing
laiguokun/Funnel-Transformer (github.com)