Sparsifying Transformer Models with Differentiable Representation Pooling

Michał Pietruszka, Łukasz Borchmann, Filip Grałński
Applica.ai, Warsaw, Poland
{michal.pietruszka, lukasz.borchmann, filip.gralinski}@applica.ai

Abstract
We propose a novel method to sparsify attention in the Transformer model by learning to select the most-informative token representations, thus leveraging the model’s information bottleneck with twofold strength. A careful analysis shows that the contextualization of encoded representations in our model is significantly more effective than in the original Transformer. We achieve a notable reduction in memory usage due to an improved differentiable top-k operator, making the model suitable to process long documents, as shown on an example of a summarization task.

Introduction
Introduction of Transformer architecture led to an immense improvement in the performance of Natural Language Processing systems (Vaswani et al. 2017; Radford 2018; Devlin et al. 2019). Nevertheless, the underlying attention mechanism is marked by the original sin of quadratic memory complexity w.r.t. the input sequence length. The vast amount of subsequent research was dedicated to overcome the mentioned drawback and make the processing of full-length documents possible (Dai et al. 2019; Beltagy, Peters, and Cohan 2020; Kitaev, Kaiser, and Levskaya 2020; Tay et al. 2020; Zaheer et al. 2020).

Recently proposed solutions limit memory by approximating the self-attention mechanism by a low-rank matrix or computing only the attention matrix’s subsets. We propose another approach for optimizing the Transformer memory complexity. It is based on the cognitively-supported hypothesis that it is possible to provide a valid answer with only selected passages of the input text available for some problems involving natural language. These passages may be of substantially shorter length than the original text. Observation of humans reading and highlighting parts of text for further analysis or synthesis led us to develop a trainable representation selection mechanism, working in such a way that human behavior is imitated.

Optimization of the self-attention complexity is achieved by learning to select encoded representations for the given task and promoting only the chosen ones to the next layer of the model. This results in significantly lower memory consumption. Here, however, the top-k selection problem arises, which is that the selection operation has to be differentiable w.r.t. the selection criterion, such as the representation scoring function. We provide a robust, high-performance solution, suitable for large k.

We demonstrate this idea’s applicability by tackling the long document summarization problem. The proposed end-to-end model is a significant improvement over the previous systems, where the extractive model was trained independently of the abstractive one.

Contribution. The specific contributions of this paper are the following: (1) We propose and validate the method to sparsify Neural Networks in a novel, previously unrecognized way. It works by learning to select the subset of best representations depending on the advantage they give on a downstream task. (2) Additionally, to the best of our knowledge, we are the first to optimize the decoder’s cross-attention complexity, which may be beneficial for inference and improve beam search with larger beam-size or allow inference with larger batch size. (3) We introduce an additional information bottleneck to the Transformer’s encoder-decoder architecture that leads to a better contextualization of representations. (4) We provide two Transformer-based architectures, drawing from our theoretical contributions, namely the Encoder-Decoder architecture and the Pyramidion. (5) We present an elegant way to train extractive-abstractive models in an end-to-end manner with only a cross-entropy loss function. (6) We present our differentiable Successive Halving Top-K selection algorithm with better complexity than previously proposed methods.

Related Works
As our work bridges fields of attention optimization and we present a proof-of-concept evaluation on a document summarization, a review of the most-relevant previous works is presented for these two subjects separately.

Sparse Attention. The observation that the full quadratic attention matrix multiplication can be avoided was previ-
The encoder layer is followed by representation pooling. Each representation is scored, and then only those with the highest scores are passed to the decoder.

Encoding can be performed as in standard Transformer architecture on the full-length input. It is, however, possible to process the text in chunks of fixed length.

Figure 1: Transformer encoder with representation pooling applied afterwards, as in the Encoder-Pooler-Decoder architecture. Each representation is scored independently, and it is possible to process long text in blocks. Representations with the highest scores are passed through the bottleneck to the decoder. The figure depicts one-layered encoder.

Figure 1: Transformer encoder with representation pooling applied afterwards, as in the Encoder-Pooler-Decoder architecture. Each representation is scored independently, and it is possible to process long text in blocks. Representations with the highest scores are passed through the bottleneck to the decoder. The figure depicts one-layered encoder.

Figure 1: Transformer encoder with representation pooling applied afterwards, as in the Encoder-Pooler-Decoder architecture. Each representation is scored independently, and it is possible to process long text in blocks. Representations with the highest scores are passed through the bottleneck to the decoder. The figure depicts one-layered encoder.

Oursly stated in multiple works. Child et al. (2019) proposed dividing the input into smaller ‘blocks’ or other subsets. Martins and Astudillo (2016) achieved sparsification by changing Softmax to Sparsemax, where the latter may assign exactly zero probability to some elements. In Tay et al. (2020), the sparse attention performance was improved by training a meta-sorting network attending to blocks of inputs.

Wang et al. (2020) observed that the self-attention mechanism is low rank and can be efficiently approximated by projecting a signal to a lower-dimensional space first and calculating inner products that constitute attention. Beltagy, Peters, and Cohan (2020) considered long-text solutions suitable for the Language Modeling problem but inadequate for tasks that benefit from bidirectional context. Their idea is to sparsify the full self-attention matrix according to an attention pattern, specifying pairs of input locations attending to one another. It works similarly to a multi-layer CNN as the local attention builds contextual representations while the global attention creates representations of the complete sequence for prediction.

**Summarization.** The approaches to summarization break into two main strategies: abstractive and extractive. The former refers to the techniques where new sentences are being generated during the process, whereas in the latter approach a subset of the words or sentences in the provided document is selected and returned as a summary (Dong 2018).

Our take is an improvement over recently proposed two-stage hybrids that extract and paraphrase in two independent steps, using modules trained separately. Nallapati, Zhou, and Ma (2016) and Gehrmann, Deng, and Rush (2018) argue that characteristics of such models are cognitively supported since humans perform image captioning and summarization in two independent steps. Gehrmann, Deng, and Rush (2018) presented a two-stage approach with a content selector made of a sequence labeler and attentional sequence-to-sequence paraphraser.

Similarly, Subramanian et al. (2019) independently trained two distinct components: the extractive summarization model and the language model conditioned on both extracted sentences and the input document. Authors of Hsu et al. (2018) weight multiple loss functions, which allows them to train models in an end-to-end manner. Chen and Bansal (2018) propose a Reinforcement Learning agent with sentence selection action for an extractor model that enables training end-to-end systems.

**Novel Approach: Encoder-Decoder with Representation Pooling**

There are plenty of problems involving texts in natural language, for which the vast amount of input text is redundant. Consider an example of summarization where only a part of the sentences is vital to produce an accurate summary. One may think of them as highlights made by a person reading the paper in such a way that it is possible to provide a summary using only the highlighted parts.

We introduce an end-to-end mechanism that performs such highlighting by scoring representations and passing only the selected ones to the next layer of the neural network.

The role of highlighting is to reduce data resolution in a roughly similar way to how pooling works in CNNs, where the original feature map is downsampled and only the most informative activations are retained. When pooling in a differentiable manner at the bottleneck of the encoder-decoder, it impacts the encoding process because the additional, orthogonal, informational bottleneck forces the model to compress more context into one representation vector of constant-length, making more efficient use of the already provided capacity.

**Architecture Outline**

Let $n$ denote number of input tokens that are projected into $d$ dimensions, resulting in a matrix of embedding representations $E \in \mathbb{R}^{n \times d}$. We define a scoring function $S_E : \mathbb{R}^d \rightarrow \mathbb{R}$ that assigns a scalar $v_i$ to each representation.
vector $E_i \in \mathbb{R}^d$. The more useful the representation is for further layers, and our training objective, the larger the value of its associated scalar is. Next, from a set of $n$ embeddings we select a subset of $k$ with our soft top-$k$ operator $I$ based on the scores in $v$ such that $I: \mathbb{R}^{n \times d} \times \mathbb{R}^n \rightarrow \mathbb{R}^{k \times d}$. The selected $k$ representations form the input for the next network layer.

**Flavors.** We consider two architectures in this work: with single or multiple pooling layers (Figure 2). Specifically, the latter is a generalization of the former to any given number of pooling layers. We use the term Encoder-Pooler-Decoder when a single pooling layer is placed after the encoder. This setup directly limits the amount of information passed to the decoder through the network’s bottleneck.

However, pooling can be applied between any subsequent layers, such that multiple operations of this type will be used in the network, introducing the bottleneck gradually along the encoding process. As a result, the same model bottleneck size can be achieved. Moreover, the decision to pool earlier has an advantage of attaining more substantial memory complexity reduction. This model will be referred to as the Pyramidion.

**Blockwise attention.** When propagating through layers, we use blockwise attention and split input into non-overlapping chunks, in such a way that the full quadratic attention is computed for each chunk. The score is then determined for each representation vector, and after selecting with the top-k operator, chosen representations are passed to the next layer. We assure our top-k operator selects representations without permuting their order, keeping them in line with their original position; thus, they are not randomly mixed or swapped, which is crucial for the next layer blockwise attention in the case of the Pyramidion.

**Scoring Functions**

Multiple scoring functions can be proposed as $S_E$. We use the following in our experiments.

**Linear.** The most straightforward scoring function is a linear transformation $S_E = Ew_T + b$ as used in conventional token classification.

**Nonlinear.** A quite natural next step is to include nonlinearity. We follow the specification of RoBERTa classification head (Liu et al. 2019), defined as $S_E = \tanh(Ew_1^T + b_1) \cdot w_2^T + b_2$.

**Attention-based.** Column-wise sum over attention matrix $A = \text{Attn}(E)$ from a specified head, that is $S_E = \sum_{j=1}^{n} A_{i,j}$.

**Embedding-based.** Based on a specified dimension in encoded space, such as $S_E = E_{*,c}$ where $c$ is a constant index.

**Comparison to Existing Sparsification Techniques**

Our method is similar to Sparse Transformer (Child et al. 2019) as we consume blocks of tokens. However, we do not experiment with this pattern and use it to encode representations that are to be pooled. The Sinkhorn Transformer is similar, as it uses a differentiable system to learn about the blocks’ importance. However, we score each representation independently and perform a differentiable selection rather than sorting.

We reduce the representation matrix’s size $E \in \mathbb{R}^{n \times d}$, but not by optimizing the size of representation dimensionality…
indicate this paper’s contributions. The total memory complexity of the vanilla Transformer depends on the number of layers \( l \), the number of tokens in the input \( n \) and the number of tokens each attends to \( n_t \). Likewise, the decoder’s cross-attention depends on the number of layers \( l \), target length \( t \), and the number of encoded tokens \( n \). The \( m \) denotes either the block-attention chunk’s size or the number of tokens one can attend to, resulting from the allowed window size. The \( r \) is a rank of the factorization matrix, which can be a constant independent of \( n \). Similarly, the number of best task-specific representations \( k \), selected after encoding, is independent of \( n \). The \( c \) is an effective number of layers in a hierarchically decreasing encoder of the Pyramidion. We evaluate the Pyramidion with \( c \) as low as 2.

### Table 1: Memory complexity of the Transformer models.

| Model                | Encoder self-attention | Decoder cross-attention |
|----------------------|------------------------|-------------------------|
| vanilla Transformer  | \( l \times n \times n \times d \) | \( l \times t \times n \times d \) |
| Sparse Transformer   | \( l \times m \times n \times d \) | \( l \times t \times n \times d \) |
| Linformer            | \( l \times n \times r \times d \) | \( l \times t \times n \times d \) |
| Longformer           | \( l \times m \times n \times d \) | \( l \times t \times n \times d \) |
| Encoder-Pooler-Decoder | \( l \times m \times n \times d \) | \( l \times t \times k \times d \) |
| Pyramidion           | \( c \times m \times n \times d \) | \( l \times t \times k \times d \) |

We narrow down the size of the representation on the output of each chosen layer, leading to the exponential reduction of memory consumption as the encoding proceeds. For example, when pooling after every layer is considered, the total memory complexity across layers would be:

\[
\sum_{i=0}^{p} 2^{-i}mnd = (2 - k/n)mnd
\]  

Where \( p \) denotes the number of passes \( p = \log_2(n/k) \).

Hence, the effective complexity of all layers is lower than \( 2mnd \), which means it is lower than 2 times the complexity of the full-size first layer.

For the decoder cross-attention, the number of input representations that target tokens can attend to is limited by \( k \), thus decreasing the memory complexity of cross attention from \( O(t \times n) \) to \( O(t \times k) \).

Optimization over quadratic sentence-length complexity is even more powerful and needed on the decoder side, as \( O(t \times n) \) complexity hurts performance of real-world applications based on auto-regressive decoding.

The blockwise attention itself reduces encoder complexity proportionally to the number of chunks. We further reduce the decoder layer’s complexity in Encoder-Pooler-Decoder models by a factor of \( n/k \), thanks to representation pooling. The Pyramidion we propose offers an additional improvement on the encoder side, such as memory consumption, which is reduced twice in each of the consecutive layers compared to the Transformer featuring blockwise attention. In other words, when \( b \) denotes the number of blocks, there is a \( b/2 \) memory saving in each block after the second layer, and \( 3b/4 \) after three layers, because the beneficial impact of pooling accumulates.

Our approach is complementary to others, thus paving a new interesting avenue of potential research. It can be applied simultaneously with other improvements as representation pooling addresses a different aspect of the memory complexity problem.

### Limitations and assumptions

A scoring function \( S_E \) and a selection mechanism have to be differentiable to instantiate a pooler. We proposed various valid functions for scoring such as linear, nonlinear, attention-based, random, and index-based. The choice of the selector operator, however, is more challenging. Specifically, a selection operation that chooses \( k \) vectors must be at least partially differentiable w.r.t. the scores these vectors obtained. In the next section, we propose a mechanism fulfilling these requirements.

#### Novel Differentiable Top-k Operator

The crux of our approach is the Successive Halving Top-K selection mechanism that finds a subset of \( k \) vector representations from \( E \) that achieved the highest value in a score vector \( v \) (as shown in Algorithm 1). In short, we perform a tournament soft selection, where candidate vectors are compared in pairs \((i,j)\), until only \( k \) remained. After each tournament’s round a new \( E' \) and \( v' \) are composed as a linear
In contrast, we always perform softmax over a pair of values, guaranteeing that there will be a candidate with a probability greater than 0.5 assigned. After each pass, the best scoring \( k \) vectors with a small noise are obtained. It is a result of interpolating with the lower-scoring element from each pair.

As stated before, we ensure that strong candidates have weakly-scoring opponents, strengthening their presence in the tournament’s next round. The fundamental requirement of this strengthening trick is to sort inputs, resulting in an additional cost of \( O(n \log(n)) \). However, in the case of modern CPUs, this cost is practically negligible. Nevertheless, the sorting step can be omitted, leading to a slightly degraded top-k approximation. During the process, a vector with considerable noise may be produced for elements with indexes closer to the \( n/2 \). Nevertheless, some noise itself is desired, as it allows gradients to propagate to elements out of the top-k.

**Evaluation**

We assess the value of our approach on a summarization task and train the models from scratch on the arXiv dataset proposed by Cohan et al. (2018), constructed assuming the goal of generating an abstract given a long scientific article.

All experiments and benchmarks were performed on a DGX-A100 server equipped with eight NVIDIA Tesla A100 GPUs. The models were trained using the Adam optimizer and cross-entropy loss, with hyperparameters specified in

\[ \sum_{i} w_i + \sum_{j} w_j \]

where the \( w_i, w_j \) is the result of a boosted softmax over scores \( v_i, v_j \).

Analogously, the new-round’s scores are calculated as:

\[ v_i \cdot w_i + v_j \cdot w_j \]

Weights are calculated using a BoostedSoftmax function, increasing the pairwise difference in scores between \( v_i \) and \( v_j \). Here, multiple functions can be used. For example, softmax with base greater than \( e \) or, equivalently, \( \text{softmax}(C x, C y) \) with constant \( C \gg 1 \).

One round halves the number of elements in \( E \) and \( v \). We perform it iteratively unless the size of \( E \) and \( v \) matches the chosen value of \( k \).

To improve convergence towards selecting the real top-k, it is desired to permute \( v \) and \( E \) first. In our Algorithm 1, we sort vectors in \( E \) by their scores \( v \) and then make pairs is such a way that \( i \)-th highest-scoring vector will be paired with \( (n - i + 1) \)-th highest-scoring vector, marked with \( j \)-th index. Here, a simple non-differentiable sorting operation suffices. Note that the selection of preferable permutation itself makes the process only partially differentiable.

**Relation to Previous Works.** Goyal et al. (2017) provides the most similar relaxation for beam search, where they continuously relaxed the top-k-argmax procedure by performing softmaxes iteratively \( k \) times and masking the previously extracted values. Each beam can contribute to the newly selected beam in every iteration, based on its distance to the max value. By replacing one-hot coded vectors with their expectations in a similar vein, Plötz and Roth (2018) relaxed the KNN hard top-k selection rule. Xie and Ermon (2019) replaced a sampling of \( k \) elements from the collection of items by a differentiable algorithm based on the Gumbel Trick.

Cuturi, Teboul, and Vert (2019) recovered sorting permutation, by solving optimal assignment problems. Similarly, Xie et al. (2020) parametrized the top-k operator as an optimal transport problem, that returns top-k elements from a given input set.

**Algorithm 1**

**Successive Halving Top-K Selection**

1: procedure TopK(\( E, v \))
2: for \( i \leftarrow 1, \log_2(\lceil n/k \rceil) \) do
3: \( E, v \leftarrow \text{SORT}(E, v) \)
4: \( E, v \leftarrow \text{Tournament}(E, v) \)
5: end for
6: return \( E \)
7: end procedure

8: procedure Sort(\( E, v \))
9: \( u' \leftarrow (v_1, v_2, \ldots) \), where \( v_i \geq v_{i+1} \) and \( v_i \in v \)
10: \( E' \leftarrow (E_1, E_2, \ldots) \), where \( v_i \geq v_{i+1} \) and \( v_i \in v \)
11: return \( E', v' \)
12: end procedure

13: procedure Tournament(\( E, v \))
14: \( n \leftarrow \frac{1}{2} ||v|| \) \hfill \( \triangleright \) Target size
15: \( d \leftarrow ||E_{n,1}|| \) \hfill \( \triangleright \) Representation depth
16: \( u' \leftarrow 0_{n,1} \)
17: \( E' \leftarrow 0_{n,d} \)
18: for \( i \leftarrow 1, n \) do
19: \( w \leftarrow \text{BoostedSoftmax}(v_i, v_{2n-i+1}) \)
20: \( E'_i \leftarrow E_i \cdot w_0 + E_{2n-i+1} \cdot w_1 \)
21: \( v'_i \leftarrow v_i \cdot w_0 + v_{2n-i+1} \cdot w_1 \)
22: end for
23: return \( E', v' \)
24: end procedure

25: return \( E', v' \)
26: end procedure
In this paper, we propose a method to render belief-propagation practical using a high-order (@xmath0) model, and use it for the task of image inpainting. By using a nonparametric prior, we avoid the need to discretize images, resulting in much smaller messages being passed between cliques. Our experiments show that belief-propagation techniques are able to produce competitive results after only a single iteration, rendering them faster than many gradient-based approaches.

In this paper, we use belief-propagation techniques to develop fast algorithms for image inpainting. Unlike traditional gradient-based approaches, which may require many iterations to converge, our techniques achieve competitive results after only a few iterations. On the other hand, while belief-propagation techniques are often unable to deal with high-order models due to the explosion in the size of messages, we avoid this problem by approximating our high-order prior model using a gaussian mixture. By using such an approximation, we are able to inpaint images quickly while at the same time retaining good visual results.

**Table 2**: Our results on arXiv summarization dataset (Cohan et al. 2018). Bold indicates the best scores for each type of model. *Layer sizes* reports each layers input length with encoder and decoder separated by an arrow; in the case of multiple consecutive layers of the same length, their exact number was presented in superscript. Length were given as multiples of 1024, e.g., $5 \times 2 \rightarrow 5 \times 2$ informs that encoder and decoder consisted of two layers consuming 512 tokens each. *Depth* presents number of layers in the encoder and decoder (separated by $\times$ character). CI denote 95% bootstrap confidence intervals of an estimate of the data (Calmettes, Drummond, and Fowler 2012). For each model, the memory *Complexity* measured as a consumption of blocks of 512 is reported, where $t$ stands for the number of chunks one can make out of the target length.

| Model           | Layer sizes [K] | Depth | ROUGE-1 (CI) | ROUGE-2 (CI) | Complexity |
|-----------------|-----------------|-------|--------------|--------------|------------|
| **Vanilla Transformer** | $5 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 26.5 (26.3 – 26.8) | 8.0 (7.8 – 8.1) | $2 + 2t$ |
|                 | $2 \times 2 \rightarrow 2 \times 2$ | 2 x 2 | 36.1 (35.8 – 36.4) | 13.2 (13.0 – 13.4) | $32 + 8t$ |
|                 | $8 \times 2 \rightarrow 8 \times 2$ | 2 x 2 | **39.6** (39.3 – 39.8) | **15.2** (14.9 – 15.4) | 512 + 32t |
| **Encoder-Pooler** |                |       |              |              |            |
| Linear | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 36.0 (35.8 – 36.3) | 13.3 (13.1 – 13.5) | $8 + 2t$ |
| Nonlinear | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 34.8 (34.5 – 35.0) | 12.7 (12.6 – 12.9) | $8 + 2t$ |
| Attention-based | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 29.0 (28.7 – 29.2) | 9.6 (9.5 – 9.8) | $8 + 2t$ |
| Random | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 32.3 (32.1 – 32.5) | 11.4 (11.3 – 11.6) | $8 + 2t$ |
| Index-based | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 35.6 (35.4 – 35.9) | 12.8 (12.7 – 13.0) | $8 + 2t$ |
| Embedding-based | $2 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 34.9 (34.7 – 35.2) | 12.7 (12.5 – 12.9) | $8 + 2t$ |
| Linear | $8 \times 2 \rightarrow 5 \times 2$ | 2 x 2 | 38.6 (38.4 – 38.8) | 15.1 (14.9 – 15.4) | 32 + 2t |
| Linear | $8 \times 2 \rightarrow 2 \times 2$ | 2 x 2 | 38.7 (38.4 – 39.0) | 15.1 (14.9 – 15.3) | 32 + 8t |
| Linear | $8 \times 2 \rightarrow 2 \times 2$ | 3 x 2 | **39.5** (39.2 – 39.7) | **15.5** (15.3 – 15.7) | 48 + 8t |
| Linear | $8 \times 2 \rightarrow 2 \times 4$ | 2 x 4 | 38.1 (37.9 – 38.4) | 15.0 (14.8 – 15.3) | 32 + 16t |
| **Pyramidion** |                |       |              |              |            |
| 8, 2 → 5×2 | 2 x 2 | 31.7 (31.5 – 32.0) | 10.7 (10.5 – 10.9) | 20 + 2t |
| 8×2, 2 → 5×2 | 3 x 2 | 38.7 (38.5 – 38.9) | 15.2 (15.0 – 15.4) | 36 + 2t |
| 8×2, 2 → 125×2 | 4 x 2 | 36.5 (36.2 – 36.7) | 14.1 (13.9 – 14.3) | 37 + 0.5t |
| 8×2, 4 → 2×2 | 3 x 2 | **39.4** (39.1 – 39.6) | **15.7** (15.5 – 16.0) | 40 + 8t |

Table 3: Output of our best-performing Pyramidion model given the McAuley and Caetano (2007) article as an input.
midions performed significantly better than their vanilla sizes. Here, all Encoder-Pooler-Decoder models and Pyra-

| Hparam          | Value |
|-----------------|-------|
| Dropouts        | .1    |
| Activation      | ReLU  |
| Emb dim         | 512   |
| FFN emb dim     | 2048  |
| Encoder positional emb | sinusoidal |
| Decoder positional emb | None |
| Batch size      | 256   |
| Learning rate   | 5e-4  |
| Learning rate decay | –     |
| Shared emb      | True  |
| Weight decay    | .1    |
| Attention heads | 8     |

Table 4: Hyperparameters used in the summarization experiments. Number of layers was model-dependent and is reported in Table 2.

Table 4. Validation was performed every three epochs and the training stopped when no progress was observed taking the seven last scores into account.

In the case of input chunking and use of blockwise attention, positions were calculated originating at the beginning of document. No positional embeddings were used on the decoder side, which is intended for algorithm simplification. On the other hand, embeddings passed down have already sufficient positional information from the encoder.

In addition to the results, the complexity will be reported as the model’s memory consumption measured in the units corresponding to the memory requirement of blocks consisting of 512 elements. Consider the toy example from Figure 2, where the block were assumed to be of size 2. Here, the vanilla Transformer had the complexity of 4·4 blocks per layer, resulting in the complexity of 48. The decoder’s complexity is 4·t, where t stands for target size. Consequently, the total complexity of the model is equal to 48 + 4·t.

Results. Table 2 summarizes results achieved by particular model configurations. We confirmed that it is possible to achieve scores indistinguishable from baseline, significantly reducing memory consumption using the representation pooling mechanism.

An increase of the length of consumed input leads to improvement in terms of ROUGE-1 and ROUGE-2 scores. However, it results in quadratic demand on memory in the case of the vanilla Transformer model. In contrast, by using blockwise attention and representation pooling, this requirement is relaxed.

For example, our 8×2→2×2 Encoder-Pooler-Decoder achieves 38.7 ROUGE-1 and 15.1 ROUGE-2, only 0.9 and 0.1 lower than the 8×2→8×2 vanilla Transformer, while consuming exactly as much memory as the 2×2→2×2 vanilla Transformer.

The validity of pooling mechanism was confirmed by series of experiments with bottlenecks of both 512 and 2K sizes. Here, all Encoder-Pooler-Decoder models and Pyramidions performed significantly better than their vanilla counterparts. The Encoder-Pooler-Decoder performs on par with baselines when challenged with the same input lengths of 2K or 8K, consuming as low as 1/16 of memory for both encoder and decoder.

From all scoring functions validated in the Encoder-Pooler-Decoder setting, the linear achieves the best results. Interestingly, the index-based selection leads to only slightly worse performance, which can be explained by the fact the model learned to compress representations from the neighborhood into a fixed-size vector.

We conducted subsequent experiments using the linear scorer and noticed that increasing the encoder length leads to further performance gains. Significantly, further reduction in the bottleneck’s size in both Encoder-Pooler-Decoder and Pyramidion does not decrease performance.

Summary

We proposed the representation pooling as a method to reduce the complexity of Transformer encoder-decoder models. Specifically, we optimized it’s self-attention and are the first to optimize the decoder’s cross-attention complexity.

Two approaches to pooling have been proposed. The decoder layer’s complexity in our Encoder-Pooler-Decoder model is reduced by a factor of n/k where n denotes the input length and k stands for the number of representations promoted to the decoder. The Pyramidion we propose offers an additional improvement on the encoder side. Here the memory consumption is reduced twice in each of the consecutive layers improving over the Transformer featuring blockwise attention. Noteworthy, our method can be applied simultaneously with existing sparsification techniques because it addresses a different aspect of the problem.

Since the process of reducing the number of representations passed down to the next layer is at the heart of the pooling operation, part of our work focused on the Top-K algorithms applicable to gradient-based optimization. We introduced a novel method of input’s successive halving through a tournament to tackle the problem. It is an improvement over existing approaches in terms of both computational complexity and approximation quality.

Applicability of both representation pooling and Successive Halving Top-K was demonstrated by solving a long document summarization task. We confirmed that using the representation pooling mechanism, one can achieve equally good results with significantly reduced memory consumption.

Moreover, the proposed end-to-end model is a significant theoretical and practical improvement over the previous systems, where the extractive model was trained independently of the abstractive one. In contrast, our mechanism does not require the introduction of additional training objective nor stage. Moreover, the pooled representations are contextualized, which significantly differentiates our approach from other extractive-abstractive models that train components independently. It is because the passed-down hidden states are calculated by attending to multiple parts of the input sentence.

We believe that the presented approach can be applied
straightforwardly in other tasks involving long document processing.

References

Beltagy, I.; Peters, M. E.; and Cohan, A. 2020. Longformer: The Long-Document Transformer. ArXiv abs/2004.05150.

Calmettes, G.; Drummond, G. B.; and Vowler, S. L. 2012. Making do with what we have: use your bootstraps. The Journal of Physiology 590(15):3403–3406.

Chen, Y.-C., and Bansal, M. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting.

Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating long sequences with sparse transformers.

Cohan, A.; Dernoncourt, F.; Kim, D. S.; Bui, T.; Kim, S.; Chang, W.; and Goharian, N. 2018. A discourse-aware attention model for abstractive summarization of long documents. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), 615–621. New Orleans, Louisiana: Association for Computational Linguistics.

Cuturi, M.; Teboul, O.; and Vert, J.-P. 2019. Differentiable ranking and sorting using optimal transport. In Wallach, H., Larochelle, H., Beygelzimer, A.; dÁlché-Buc, F.; Fox, E.; and Garnett, R., eds., Advances in Neural Information Processing Systems 32. Curran Associates, Inc. 6861–6871.

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J. G.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. ArXiv abs/1901.02860.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In NAACL-HLT.

Dong, Y. 2018. A survey on neural network-based summarization methods. ArXiv abs/1804.04589.

Gehrmann, S.; Deng, Y.; and Rush, A. M. 2018. Bottom-up abstractive summarization.

Goyal, K.; Neubig, G.; Dyer, C.; and Berg-Kirkpatrick, T. 2017. A continuous relaxation of beam search for end-to-end training of neural sequence models.

Hsu, W.-T.; Lin, C.-K.; Lee, M.-Y.; Min, K.; Tang, J.; and Sun, M. 2018. A unified model for extractive and abstractive summarization using inconsistency loss.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2020. Reformer: The Efficient Transformer. ArXiv abs/2001.04451.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Martins, A. F. T., and Astudillo, R. F. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification.

McAuley, J. J., and Caetano, T. S. 2007. High-order non-parametric belief-propagation for fast image inpainting.

Nallapati, R.; Zhou, B.; and Ma, M. 2016. Classify or select: Neural architectures for extractive document summarization.

Plütz, T., and Roth, S. 2018. Neural nearest neighbors networks.

Radford, A. 2018. Improving language understanding by generative pre-training.

Subramanian, S.; Li, R.; Pilault, J.; and Pal, C. 2019. On extractive and abstractive neural document summarization with transformer language models.

Tay, Y.; Bahri, D.; Yang, L.; Metzler, D.; and Juan, D.-C. 2020. Sparse Sinkhorn Attention.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. ArXiv abs/1706.03762.

Wang, S.; Li, B. Z.; Khabsa, M.; Fang, H.; and Ma, H. 2020. Linformer: Self-attention with linear complexity.

Xie, S. M., and Ermon, S. 2019. Reparameterizable subset sampling via continuous relaxations.

Xie, Y.; Dai, H.; Chen, M.; Dai, B.; Zhao, T.; Zha, H.; Wei, W.; and Pfister, T. 2020. Differentiable top-k operator with optimal transport.

Zaheer, M.; Guruganesh, G.; Dubey, A.; Ainslie, J.; Alberti, C.; Ontanón, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; and Ahmed, A. 2020. Big Bird: Transformers for Longer Sequences.