TransformerFTC: Scaling Low-Dimensional Transformers for Higher Performance

Viswesh Krishna
Department of Computer Science
Stanford University
viswesh@stanford.edu

Abstract

Transformers have achieved high performance in neural machine translation (NMT), with *deep* models—those with numerous layers—being the most successful. However, such models are not only slow to train but are also computationally expensive. In this work, we present TRANSFORMERFTC, a Transformer architecture that utilizes subsampling methods which allow for training deep Transformers at a lower computational cost. Specifically, the Transformer layers are divided into a series of *blocks*, with each block operating over a pooled sequence of shorter length (*time compression*) and/or of a lower feature dimension (*feature compression*). Validating on the IWSLT'14 German-English dataset, we achieve performance equivalent to a vanilla Transformer with a 25% training speedup. Our results suggest that compressing the upper layers of a Transformer are a promising strategy for model efficiency.

1 Introduction

Transformers have been hugely successful in many NLP tasks [Vaswani et al., 2017, Devlin et al., 2018], including machine translation. Further, increasing pretraining duration and using larger models have caused continued improvement in performance for neural self-attention models. However, currently it is still extremely expensive to train state of the art self-attention models which has been a major factor in limiting adoption.

Previous work has demonstrated that *deep* networks, which have a larger number of layers achieve higher performance than their *wide* counterparts, which use a higher hidden state size. However, Transformers generally require a certain minimum feature dimension (typically 256 or 512) to achieve strong performance. This means that in practice, the depth of Transformer networks is limited, due to both runtime and memory constraints.

In this work, we propose TRANSFORMERFTC, a scaling lower-dimension architecture that allows the construction of extremely deep networks without excessive memory or gradient cost. Taking inspiration from Dai et al. [2020], we propose the use of sequential *blocks* of Transformer layers. Each layer has a progressively decreasing feature and/or time dimension, ameliorating high memory use. In order to capture the full informational complexity of the input, we combine the output of the first block using an upscaling function. We consider this as separating two of the important aspects of translation: word-level translation (output of first block) and higher-order syntactic operations (deep Transformer). We achieve promising results with feature-only and feature and time scaling Transformers.

CS230: Deep Learning, Autumn 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related Work

There have been many approaches to reduce costs of pretraining and finetuning attention models. Previous work has attempted to optimize self attention by either replacing it with less expensive operations or approximating it effectively. Wu et al. [2019] showed that a linear time convolution-like operator can actually exceed the performance of the quadratic-time self-attention layer in the Transformer at sentence-to-sentence translation. Wang et al. [2020] show that quadratic self-attention can be be represented as low rank matrices thereby imitating linear time self-attention which performs on par with standard Transformers.

Dai et al. [2019] proposed TransformerXL which learns beyond fixed length contexts. They propose a novel relative positional embedding scheme - an idea adapted from Shaw et al. [2018] - which they see outperforms the Transformer's original absolute positional system. Further, they reuse previous activations which enables faster computation.

Prior work has also looked at compressing different aspects in the standard Transformer. Rae et al. [2019] propose Compressive Transformer which compresses past memories for learning long-range sequences. Khetan and Karnin [2020] reduced the number of parameters to produce a lighter and more efficient BERT. Model distillation for BERT as proposed in Sanh et al. [2019] and sequence-level knowledge distillation as proposed in Kim and Rush [2016] have also been effective in reducing the number of parameters.

In order to improve efficiency, Dai et al. [2020] proposed Funnel-Transformer which removes the redundant full-length token representations used in the hidden states of the standard Transformer. Funnel-Transformer gradually reduces the sequence length of the hidden states at deeper layers through successive pooling along the sequence axis thus leading to a much smaller representation than the encoded input. Dai et al. [2020] also make use of relative multihead self-attention as proposed in Transformer-XL [Dai et al., 2019] to cache previous computed activations and learn longer sequence contexts. The compression along sequence length and the use of relative positional embedding saves computation cost which is allocated for training larger or deeper models.

Our model is inspired by the architecture of the Funnel Transformer. However, instead of only pooling along the sequence length as in Funnel-Transformer, we pool along both time and feature axes to achieve greater compression.

3 Methods

3.1 Background

Transformer Architecture The Transformer proposed by Vaswani et al. [2017] is a high capacity neural network consisting of a set of Transformer encoder and decoder layers. Each Transformer layer consists of two modules - multi-head Self-Attention (S-Attn) and Position-wise Feed Forward Network (P-FFN) - with added residual connections and layer normalizations. Particularly, given an input of length T with hidden states $\mathbf{h} = [h_1, \ldots, h_T]$, we can express a Transformer layer's computation as:

$$\mathbf{h} \rightarrow \text{LayerNorm}(\mathbf{h} + \text{S-Attn}(\text{Q=}\mathbf{h}, \text{KV=}\mathbf{h})) \quad (1)$$

$$h_i \rightarrow \text{LayerNorm}(h_i + \text{P-FFN}(h_i)) \quad \forall i \in 1, \ldots T \quad (2)$$

3.2 Proposed Architecture

We inherit high capacity of Transformer models by using a set of interleaved Self-Attention and Position-wise Feed Forward Networks with residual connections and layer normalization as described in Vaswani et al. [2017]. To achieve feature compression, we use an encoder which gradually reduces the feature dimension of the hidden states in the deeper layers. We achieve time compression by gradually reducing the sequence length in the hidden states in the deeper layers during the forward pass through the encoder. In the decoder layer, we upsample along both the feature and time dimensions to the original feature and time dimensions to reconstruct full sequence token-level representations. The decoder is identical to the standard Transformer decoder which uses 6 decoder layers. A high level visualization of our proposed architecture can be found in Figure

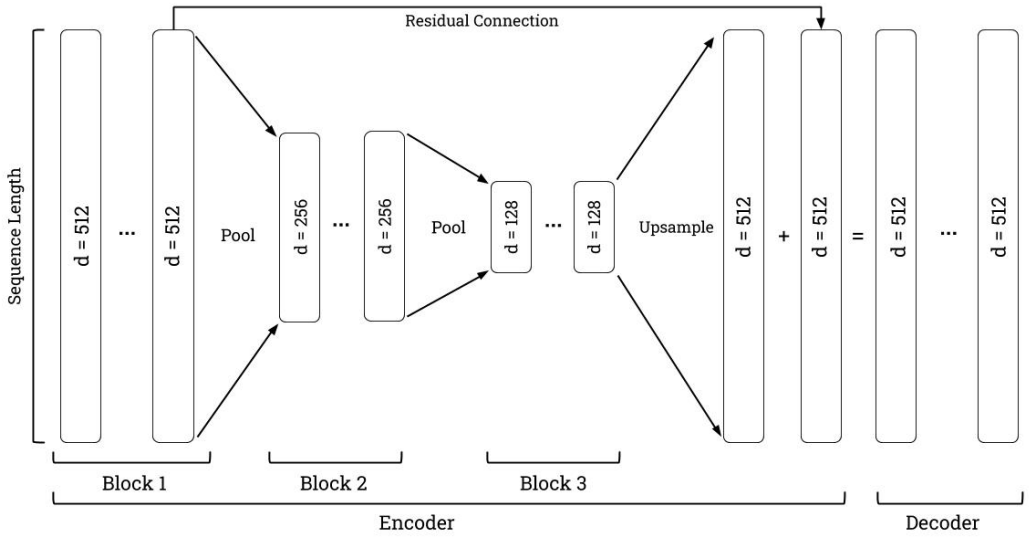

Figure 1: **High-level visualization of TRANSFORMERFTC** We compress the feature dimension d and time dimension (sequence length) gradually across hidden states in the deeper layers of the network.

3.2.1 Encoder

The encoder consists of n_{block} consecutive blocks, each with n_{layer} Transformer layers. We individually describe the implementation of feature and time compression.

Feature Compression We implement feature compression as a series of gradual downsampling steps between blocks while the feature dimension remains constant within a single block. Between each block, the feature dimension is downsampled with a pooling window of size 2, using either mean pooling or a linear downsampler. For example, if the input has a feature dimension of 512, then the first block has transformer layers with a feature dimension of 512, followed by a second block with a feature dimension of 256, and so on. In the self-attention block for the first layer after downsampling, we use a query vector with the downsampled feature dimension (e.g. 256) while using key and value vectors with the previous feature dimension (e.g. 512). This approach sidesteps the ineffective compression that relatively simple pooling operations alone would provide as described in Dai et al. [2020].

Time Compression We implement time compression in a manner analogous to feature compression. Time compression here refers to compressing the sequence length of the hidden states in the deeper layers of the network. We gradually reduce the sequence length using a series of downsampling steps between encoder blocks with the time dimension (sequence-length) remaining the same within the layers inside an encoder block. Particularly, the time dimension is downsampled with a pooling window of size 2, using a mean-based downsampler. Further, just as in feature compression, in the self-attention for the first block after downsampling we use key and value vectors with the previous time dimension (e.g. t) while using a query vector with the downsampled time dimension (e.g. t).

Reconstructing Full-Sequence Token Representations At the end of the encoder, we have a low-dimensional output in both feature and time dimensions. To preserve the rich information latent in the model's input token embeddings, we sum the upsampled output of the low-dimension final block with the full-dimension output of the first block. For example, given an input sequence of length t with embedding of size 512, we would sum the 512-dimensional output of the last layer of the first encoder block with the low-dimensional output of the final encoder block upsampled to 512 along the feature dimension and t along the time dimension which would serve as the encoder output.

Models We define four models for comparison: the standard Transformer from Vaswani et al. [2017], TRANSFORMERFC (feature compression only), TRANSFORMERTC (time compression only), and TRANSFORMERFTC (feature and time compression). Our proposed architecture enables feature and time compression to be implemented independently of each other.

3.3 Comparison

Dataset and Evaluation Metric We use the IWSLT'14 German-English dataset for comparison. We use the pre-processing script `prepare-iwslt14.sh` available in the `Fairseq` library [Ott et al., 2019] which implements preprocessing, training and evaluation functionality. This script provides 160k training sentence pairs. We use the BLEU score proposed by Papineni et al. [2002] as our primary comparison metric.

Code Code for the model is available at `https://github.com/VisweshK/fairseq`. TRANSFORMERFTC is implemented in `fairseq/models/funnel.py` and inherits from the standard `TransformerModel` architecture in Fairseq. The custom encoder layer for TRANSFORMERFTC is implemented in `fairseq/modules/funnel_layer.py`.

Baseline We compare against the standard Transformer from Vaswani et al. [2017] implemented within Fairseq. We utilize the details from `https://github.com/VisweshK/fairseq/tree/master/examples/translation` to train and evaluate a Standard 6-encoder, 6-decoder Transformer. Fairseq adapts the standard Transformer architecture from Vaswani et al. [2017] to the IWSLT'14 German-English dataset by using 1024 instead of 2048 as the encoder and decoder P-FFN embedding dimensions and 4 instead of 8 attention heads. We train TRANSFORMERFTC with the same beginning encoder layer dimension and training hyperparameters as used for the standard Transformer.

4 Results

We find that TRANSFORMERFTC outperforms the Standard Transformer with a 25% speedup. We present our comparisons with previous work in Table 1. We further compare the effect of using a pooling window of size 4 instead of 2 and the effect of using learned linear layers on feature compression in Table 2.

5 Discussion

We find that compressing the upper layers in TRANSFORMERFTC helps reduce training time by 25% while retaining the performance of the standard Transformer. Both feature and time compression independently retain performance of the standard Transformer and produce speedups of 10% each. Notably, increasing the number of encoder layers did not have a large effect on performance for either time or feature compression. We find that when combining feature and time, we achieve the best results with a speedup of 25% while outperforming the standard Transformer. The saved computations from TRANSFORMERFTC can be re-invested to improve model capacity and performance.

We observe that using a larger pooling window size of 4 instead of 2 during feature compression further reduces runtime per epoch while simultaneously increasing performance. Furthermore, we find that learned linear transformations for feature compression are not beneficial as the mean pooling itself ensures effective linear compression due to the self-attention between the lower dimensional query vector and the higher dimensional key-value vectors.

Our methods were demonstrated to speed up training significantly by reducing the computational complexity of the upper layers. However, in practice this may not improve decoding performance significantly, as decode time is roughly proportional to the $O(n)$ cost of autoregressive decoding as shown in Kasai et al. [2020]. To address this, we suggest that future work investigate moving layers from the decoder to the encoder, allowing more of the speed gains in the encoder to transfer to decode time.

Model	Blks	Layers/Blk	BLEU	Runtime
Previous Work [Gehring et al., 2016]				
STATISTICAL	–	–	28.4	–
POOLING	–	–	19.7	–
CONVOLUTIONAL	–	–	20.1	–
LSTM	–	–	27.3	–
BILSTM	–	–	29.8	–
Previous Work [Vaswani et al., 2017]				
TRANSFORMER	1	6	34.4	–
Our Work				
TRANSFORMER	1	6	34.9	216
FEATURE	2	3	34.8	195
	2	4	35.1	202
	3	3	34.7	232
	4	4	**35.2**	380
	6	4	34.9	440
TIME	2	3	35.0	196
	2	4	35.1	207
FEATURE+TIME	2	3	35.1	**163**
	2	4	35.0	181

Table 1: Comparison with previous methods. All compression models use a pooling window of size 2 with mean pooling. Runtime reported is seconds per epoch.

Model	Blks	Layers/Blk	Pooling Window	BLEU	Runtime
FEATURE	2	3	2	34.8	195
			4	**35.2**	**186**
			2 (linear)	35.1	204
	3	3	2	34.7	232
			4	35.1	212

Table 2: Comparison of Pooling Window Size and Downsampling Strategies for Feature Compression

6 Conclusion

In this work, we demonstrate that Transformer models can save 25% of the training time while retaining performance. With the proposed TRANSFORMERFTC architecture, we compress across both feature and time dimensions to save computation while maintaining performance of the standard Transformer. Next steps are to leverage this saved computation to achieve higher performance with deeper models.

7 Acknowledgements

I would like to acknowledge Ethan A. Chi (ethanchi@cs.stanford.edu) for his guidance throughout this project. I would also like to acknowledge the Stanford NLP group for access to compute resources. I would finally like to acknowledge Avoy Datta's (avoy.datta@stanford.edu) helpful comments on an early version of this report.

References

Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Z. Dai, G. Lai, Y. Yang, and Q. V. Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv preprint arXiv:2006.03236*, 2020.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

J. Kasai, N. Pappas, H. Peng, J. Cross, and N. A. Smith. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*, 2020.

A. Khetan and Z. Karnin. schubert: Optimizing elements of bert. *arXiv preprint arXiv:2005.06628*, 2020.

Y. Kim and A. M. Rush. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, and M. Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.