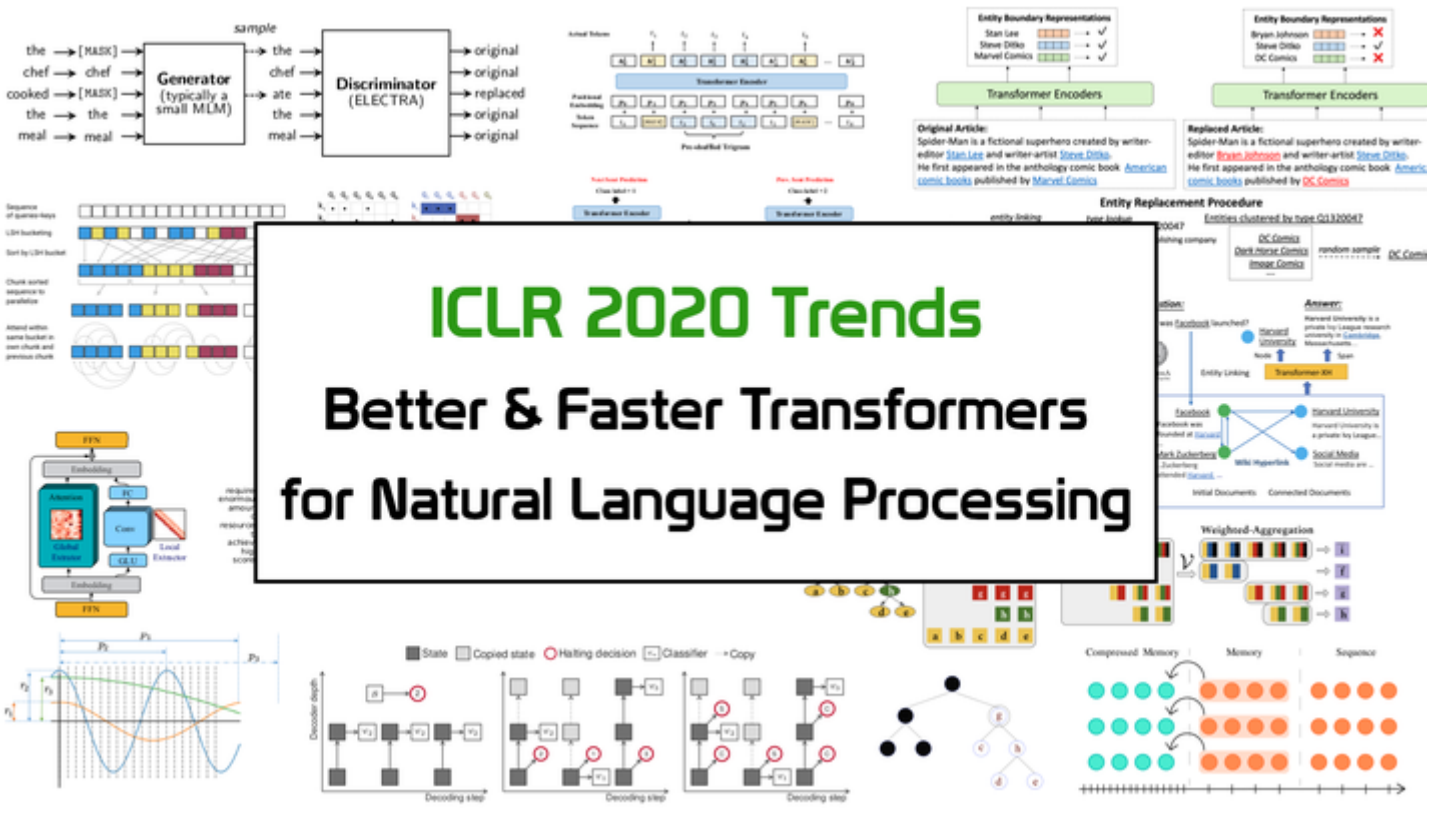
ICLR 2020 Trends: Better & Faster Transformers for Natural Language Processing

Gabriele Sarti

May 3, 2020 · 14 min read



(For an overview of the Transformer, see The Illustrated Transformer by Jay Alammar)

The Transformer architecture was first proposed in Attention is All you Need as a valid alternative to sequential language modeling approaches like LSTMs and has since then become ubiquitous in the field of Natural Language Processing, pushing the state-of-the-art in most downstream language-related tasks.

This year's edition of the International Conference on Learning Representation (ICLR) brought a lot of promising revisions to the original Transformer and its more recent variants BERT and Transformer-XL. Proposed improvements address the well-known weaknesses of Transformers, namely:

- Optimizing the self-attention computation.
- Injecting linguistically-motivated inductive biases in the model architecture.
- Making the model more parameter and data-efficient.

This post wants to summarize and provide a high-level overview of those contributions, highlighting current trends in the development of better and faster models for Natural Language Processing. All image credits go to their respective paper authors.

Index

Self-Attention Variants

Scaled dot-product self-attention is one of the main components in the standard Transformer layer, enabling the modelling of dependencies regardless of their distance in the input. The self-attention operation projects an input activation tensor $\mathbf{A}$ to queries Q of dimension d_k, keys K of dimension d_k and values V of dimension d_v, returning a weighted version of V:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \tag{1}$$

In the **multi-head self-attention** variant, the attention function is applied in parallel to h version of queries, keys and values projected with learned projections W, and outputs are finally concatenated and projected again to obtain final values:

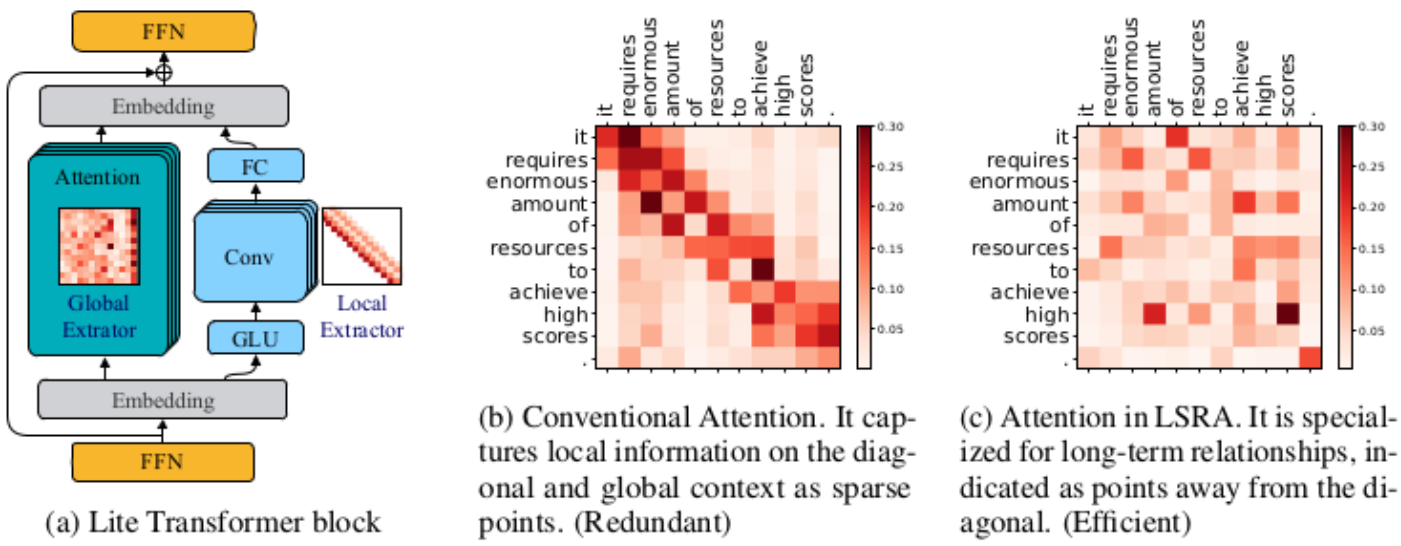
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2}$$

This section presents some variants of the self-attention component that make it more efficient and effective in the context of language applications.

Long-Short Range Attention

Introduced in: Lite Transformer with Long-Short Range Attention by Wu, Liu et al.



(a) Lite Transformer block

(b) Conventional Attention. It captures local information on the diagonal and global context as sparse points. (Redundant)

(c) Attention in LSRA. It is specialized for long-term relationships, indicated as points away from the diagonal. (Efficient)

Figure 3: Lite Transformer architecture (a) and the visualization of attention weights. Conventional attention (b) puts too much emphasis on local relationship modeling (see the diagonal structure). We specialize the local feature extraction by a convolutional branch which efficiently models the locality so that the attention branch can specialize in global feature extraction (c). More visualizations are available in Figure A1.

Conventional self-attention is deemed as redundant since it was empirically shown to put excessive emphasis on local relations inside a sentence, which can be modeled more efficiently by a standard convolution, as shown also in On the Relationship between Self-Attention and Convolutional Layers. While the redundancy may help model performances in some cases, it is not suitable for lighter applications.

Long-Short Range Attention (LSRA) makes the computation more efficient by splitting the input into two parts along channel dimensions and feeding each to two modules: a **global extractor** using standard self-attention and a **local extractor** using light depth-wise convolutions. Authors report a $2\times$ reduced overall computation for the model, making it suitable for mobile settings.

Tree-Structured Attention with Subtree Masking

Introduced in: Tree-Structured Attention with Hierarchical Accumulation by Nguyen et al.

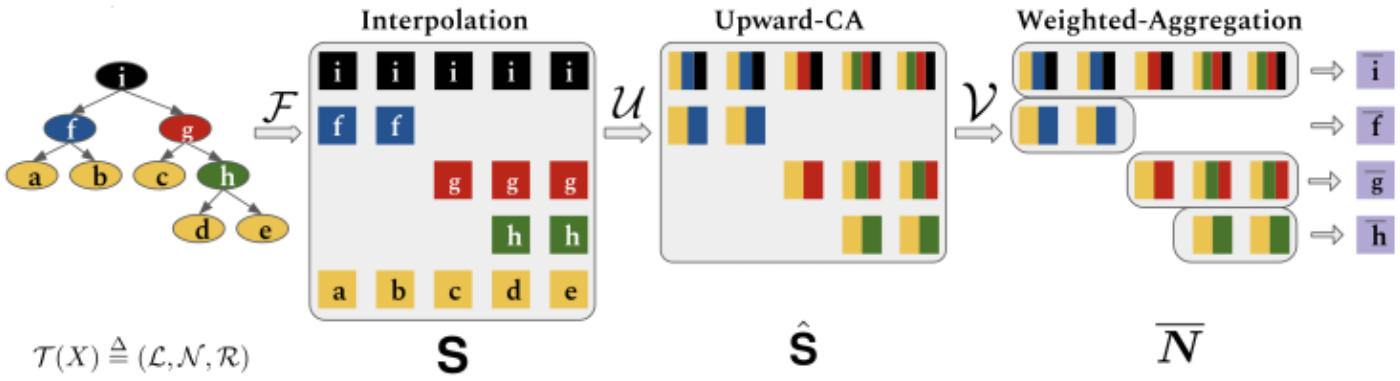


Figure 1: The hierarchical accumulation process of tree structures (best seen in colors). Given a parse tree, it is interpolated into a tensor **S**, which is then accumulated vertically from bottom to top to produce $\hat{\mathbf{S}}$. Next, the (branch-level) component representations of the nonterminal nodes are combined into one representation as $\overline{N}$ by weighted aggregation. Multi-colored blocks indicate accumulation of nodes of respective colors. The rows of **S** in Eq. 5 are counted from the bottom.

A weakness of the standard Transformer is the absence of inductive biases to account for the hierarchical structure of language. This is due in part to the difficulty in operating with tree-like structures that are usually modeled by recurrent or recursive mechanisms while maintaining the constant parallel time complexity of self-attention.

The proposed solution leverages constituency parses of input text to build a tree of hidden states, using **hierarchical accumulation** to build the value of non-terminals as the aggregation of lower representations in the tree. The final output representation is built by performing a **weighted aggregation** of branch-level representations.

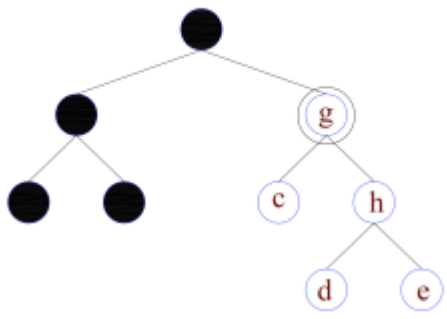


Figure 3: Subtree masking. Given the query at position g, attentions are only included within the g-rooted subtree, while the remaining elements are masked out (shaded).

An interesting addition is the use of **subtree masking** to filter out superfluous noise by constraining the attention of each node query only on its subtree descendants. The cost for this inductive bias is an increased computational and memory cost, which is then mitigated using parameter sharing

Hashed Attention

Introduced in: Reformer: The Efficient Transformer by Kitaev et al.

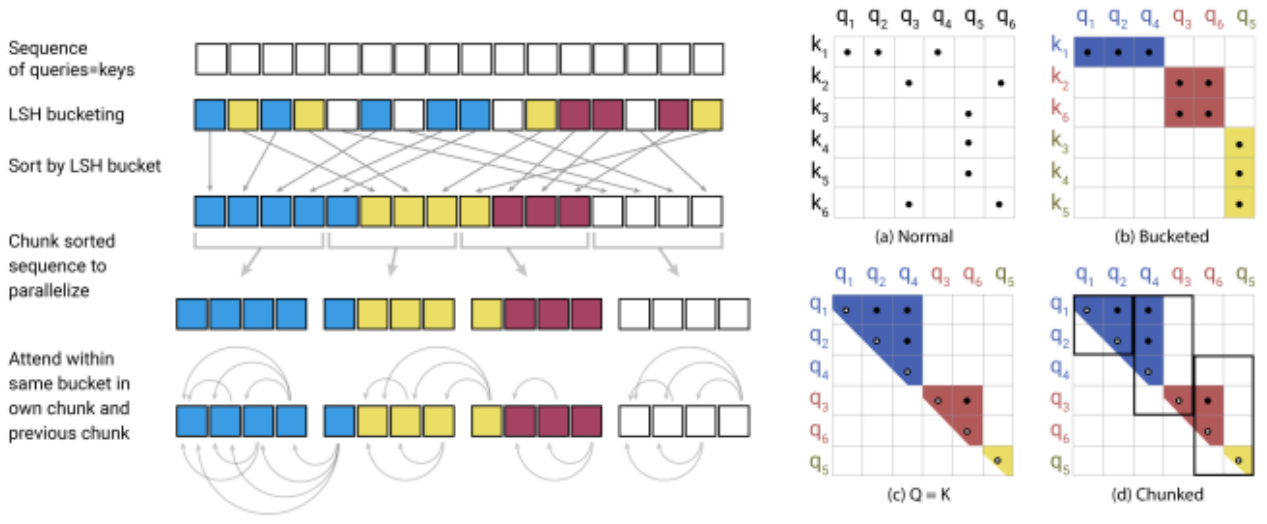


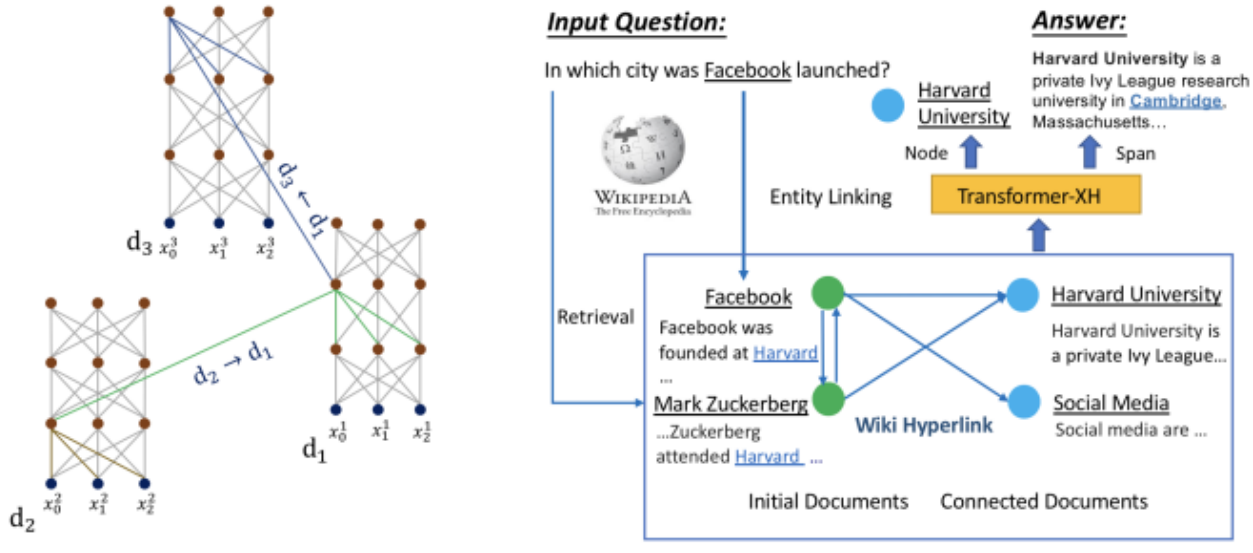
Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.

In the self-attention equation the factor QK^T represents a bottleneck, taking $\mathcal{O}(L^2)$ for input sequences of length L both in computational and memory complexity. This effectively hinders the possibility of modeling long sequences.

Reformer proposes to restrict the pool of candidates attended by each query to a small set of neighbors found through **locally-sensitive hashing**. Since LSH bucketing employs random projections, similar vectors may sometimes fall in different neighborhoods; an approach using multiple parallel rounds of hashing is suggested to mitigate this issue. Using LSH attention reduces the computational cost of the self-attention operation to $\mathcal{O}(L \log L)$, allowing the model to operate on longer sequences.

eXtra Hop Attention

Introduced in: Transformer-XH: Multi-Evidence Reasoning with eXtra Hop Attention by Zhao et al.



(a) Hop attention on the path $d_2 \rightarrow d_1 \rightarrow d_3$. (b) Transformer-XH in Multi-hop QA

Figure 1: The eXtra Hop attention in Transformer-XH (a) and its application to multi-hop QA (b).

While Transformers were optimized to operate on single sequences or pairs of sequences, they can hardly generalize to settings where evidence is scattered in multiple pieces of text, as in the challenging task of **multi-hop question answering**.

Transformer-XH introduces a new variant of attention, **eXtra Hop Attention**, that can be applied to a graph of text sequences connected by edges (e.g. hyperlinks). This new attention mechanism uses the special `[CLS]` token at the beginning of each sequence as an **attention hub** that attends to all other connected sequences in the graph. The resulting representation is then combined to the one obtained by standard self-attention through a linear projection. The resulting model shows significant improvements for tasks requiring reasoning over graphs, at the cost of the extra computations introduced by the new attention mechanism.

Training Objectives

The pre-training of Transformer models is usually achieved by the mean of multiple unsupervised objectives, leverage huge quantities of non-annotated texts. The most common tasks used for this purpose are **autoregressive language modeling**, also known as standard language modeling (LM), and **autoencoding of masked input**, often referred to as **masked language modeling (MLM)**.

The standard Transformer implementation and its GPT variants adopt the autoregressive approach, leveraging a unidirectional context (forward or backward) inside a sequence $\mathbf{x} = (x_1, \ldots, x_L)$ to estimate next token probability:

$$p(\mathbf{x}) = \prod_{l=1}^{L} p(x_l | \mathbf{x}_{<or>l})$$

Instead, BERT-like approaches use a bidirectional context to recover a small fraction of the input that was artificially replaced by special `[MASK]` tokens. This variant was shown to be especially effective for downstream natural language understanding tasks.

Besides word-level modeling, a sentence-level classification task like **next sentence prediction (NSP)** is usually added to the training procedure since many important language applications require an understanding of the relationship between two sequences.

While those tasks seem to induce meaningful token and sentence-level representation, many of the approaches covered in this section suggest better alternatives that make learning more efficient and grounded in the structure and the content of the input.

Discriminative Replacement Task

Introduced in: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators by Clark et al.

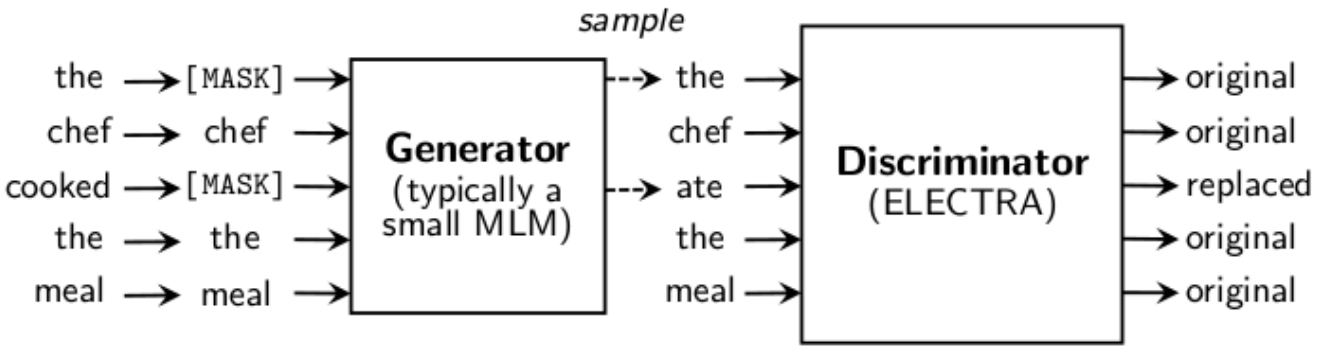


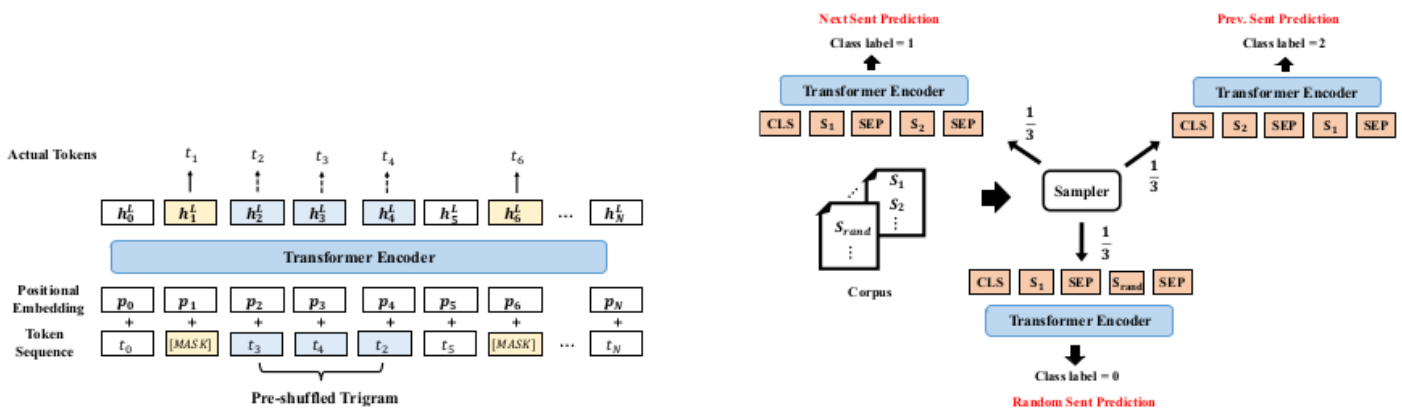
Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

The masking strategy used in BERT-like models is quite data inefficient, using only ~15% of the input text to complete the MLM task. However, the percentage of masked data can hardly be increased since having too many masked tokens may degrade the overall context information.

ELECTRA proposes a simple yet effective approach to cope with this inefficiency. A small masked language model is trained and then used as a generator to fill the masked tokens in the input with its predictions, as in normal MLM. However, the new task for the main model will be a **discriminative** one: instead of predicting masked tokens, the model has to detect which tokens have been replaced by the generator. This allows leveraging the entire input sequence for training. As mentioned by the authors, this approach consistently outperforms MLM pre-training given the same compute budget.

Word and Sentence Structural Tasks

Introduced in: StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding by Wang et al.



(a) Word Structural Objective (b) Sentence Structural Objective

Figure 1: Illustrations of the two new pre-training objectives

As seen previously, Transformers do not explicitly account for structures present in the input. While tree-structured attention injects a heavy hierarchical bias in the model architecture, **StructBERT** adopts two lighter but effective approaches to make the resulting representations more aware of the underlying sequentiality of language.

The first is a **word structural objective** where trigrams inside the inputs are randomly shuffled, and their original position must be reconstructed. This is done in parallel with normal MLM. The **sentence structural objective** is a lighter variant of the sentence reordering introduced in ERNIE 2.0 and equal to the **sentence ordering prediction** introduced in ALBERT: given a pair of sentences (S_1, S_2) as input, we ask the model to discriminate whether S_2 precedes, follows or is unrelated to S_1. This new task extends the standard NSP, which was deemed as too easy for learning meaningful sentence relations. These additions result in significant improvements over standard benchmarks for natural language understanding.

Type-Constrained Entity Replacement

Introduced in: Pretrained Encyclopedia: Weakly Supervised Knowledge-Pretrained Language Model by Xiong et al.

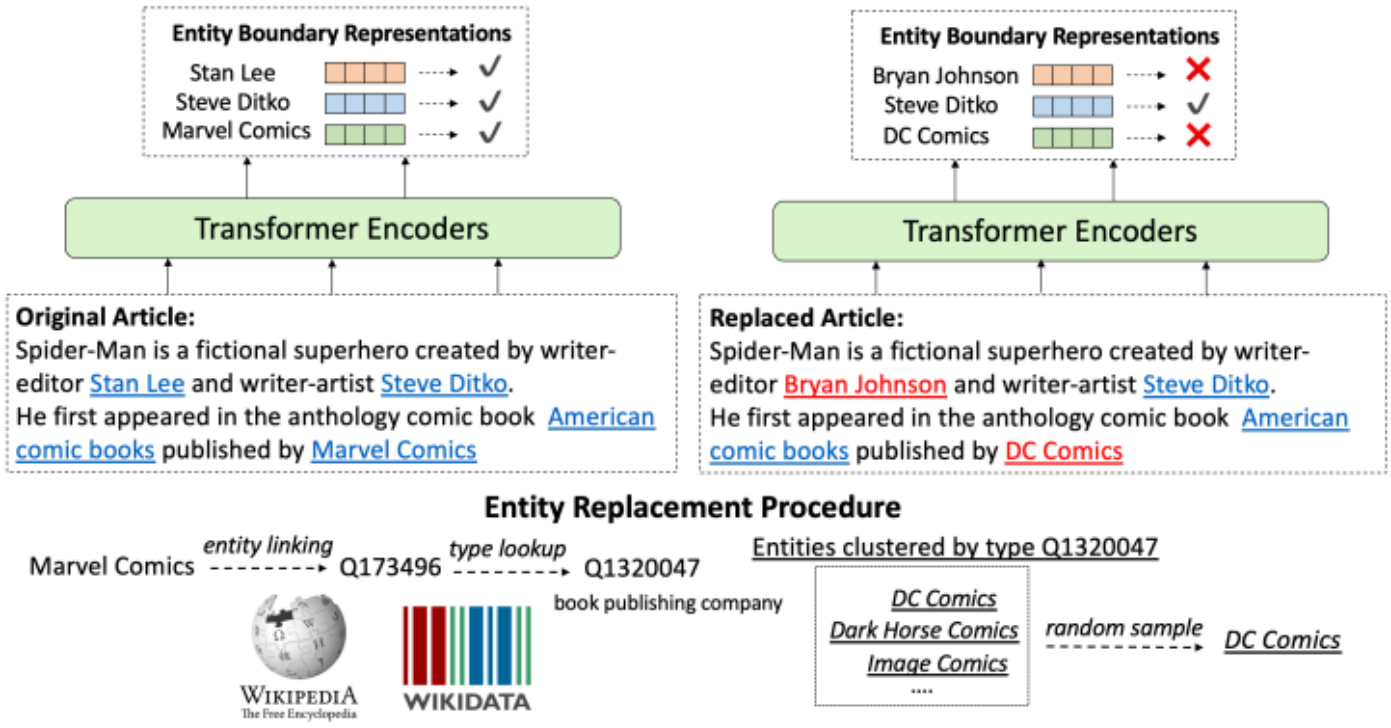


Figure 1: Type-Constrained Entity Replacements for Knowledge Learning.

While it was shown that pre-trained Transformer models implicitly capture real-world knowledge, their standard training objectives do not explicitly take into account the entity-centric information needed for robust reasoning over real-world settings.

Type-constrained entity replacement is a weakly supervised approach where random entities in the text are replaced with other entities taken from Wikidata that have the same entity type. The model then uses a discriminative objective similar to the one of ELECTRA to determine which entities were replaced. This is done along with MLM in a multi-task setup, and authors report significant improvements in settings requiring a deeper entity understanding, such as **open-domain QA** and **entity typing**.

Embeddings

The original Transformer relies on two sets of embeddings to represent the input sequence:

- Learned **word embeddings** for each token present in the vocabulary, used as token vector representations for the model.
- Fixed **positional embeddings (PE)**, used to inject information about the position of tokens in the sequence. For position pos and dimension i, those correspond to sinusoidal periodic functions that were empirically shown to perform on par with learned embeddings, and were chosen to enable extrapolation for longer sequences:

$$PE_{pos,2i} = \sin(\mathrm{pos}/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(\mathrm{pos}/10000^{2i/d_{model}})$$

For BERT-like models able to operate on multiple input segments, a third set of learned **segment embeddings** is used to differentiate tokens belonging to different sentences.

All those embeddings have the same dimensions and get summed together to obtain an input representation.

Approaches introduced in this section aim to inject more structure in the embeddings, or to optimize their dimension for better efficiency.

Position-Aware Complex Word Embeddings

Introduced in: Encoding word order in complex embeddings by Wang et al.

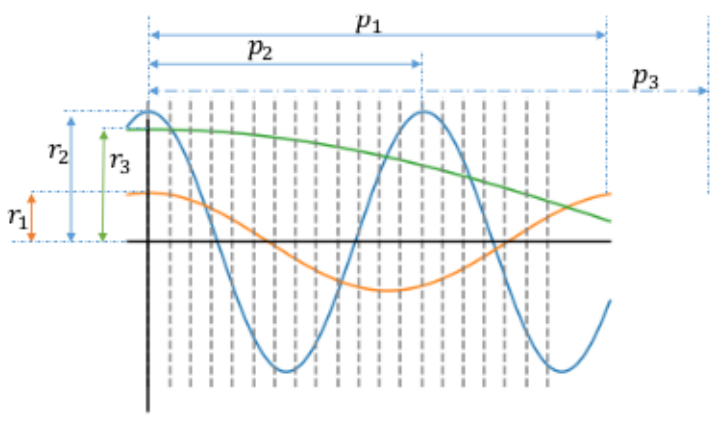


Figure 1: 3-dimensional complex embedding for a single word in different positions. The three wave functions (setting the initial phases as zero) show the real part of the embedding; the imaginary part has a $\frac{\pi}{2}$ phase difference and shows the same curves with its real-valued counterpart. The x-axis denotes the absolute position of a word and the y-axis denotes the value of each element in its word vector. Colours mark different dimensions of the embedding. The three cross points between the functions and each vertical line (corresponding to a specific position pos) represent the embedding for this word in the pos-th position.

While PE capture different positions in the input, they do not explicitly take into account the relation between those positions, i.e. ordered relationships such as adjacency or precedence. This problem was already addressed in Transformer-XL by leveraging relative distances between words instead of raw position indices.

A proposed improvement is to generalize word embeddings to continuous functions defined over positions, extending the solutions to the complex-valued domain to benefit from richer representations. The resulting **complex-valued embeddings** introduce new parameters for amplitudes, frequencies and initial phases that determine various properties of the embedding such as position sensitivity. Empirical results show that the complex embeddings with parameter-sharing schemas outperform previous embedding approaches without a significant increase in the number of trainable parameters.

Hierarchical Embeddings

Introduced in: Tree-Structured Attention with Hierarchical Accumulation by Nguyen et al.

Figure 2: Hierarchical Embeddings. Each block $\mathbf{E}_{i,j}$ is an embedding vector $[e_x^v; e_y^h]$ with indices x, y following the syntax "$x; y$", where $x = |V_j^i|$ and $y = |H_j^i|$. "**0**" indicates no embedding.

In the overview of tree-structured attention, we saw how hierarchical accumulation is used to form a representation based on descendants for nonterminal nodes. This procedure, however, has the disadvantage of not taking into account the hierarchical structure of descendants.

Hierarchical embeddings are used to inject this structural bias by concatenating **vertical** and **horizontal embeddings matrices** representing respectively hierarchical ordering inside branches and relationships between siblings nodes in a subtree. Those embeddings are shared across attention heads, thus accounting only for 0.1% of the total parameters.

Factorized Embedding Parametrization

Introduced in: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations by Lan et al.

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

In recent models based on BERT and Transformer-XL the input embeddings size E is tied with the hidden layer size H, i.e. $E \equiv H$. This is very impractical since, to augment the expressivity of hidden representations used to learn *context-dependent representation*, one should also increase the size of the embedding matrix $\mathbf{M} = V \times E$, where V is the vocabulary size. Even for relatively small hidden layer dimensions, this results in billions of parameters that are rarely updated during training.

ALBERT authors propose to insert a projection between E and V to make both dimensions independent, an approach that is especially efficient to reduce the parameter count when $H \gg E$. As a result, an ALBERT base with $E = 128$ and $H = 768$ obtains performances comparable with a BERT base with the same configuration on many downstream tasks, using 21M fewer parameters (89M in Table 3 vs 110M for BERT).

Model Architecture

The original Transformer architecture is composed of an encoder and a decoder, each composed by a stacked sequence of identical layers that transform input embeddings in outputs having the same dimension (hence the name Transformer).

Each layer of the Transformer encoder is composed of two sublayers, a multi-head self-attention mechanism and a feed-forward network, surrounded by residual connections and followed by layer normalization. The decoder includes a third layer that performs multi-head self-attention over the encoder output and modifies the original self-attention sublayer to prevent attending to future context, as required by the autoregressive language modeling objective presented above.

Bidirectional variants of the Transformer drop the decoder structure and focus solely on the encoder to generate the contextual embeddings needed for various tasks, including MLM.

Transformer-XL notably introduces a notion of **memory** for Transformer networks, where hidden states obtained in previous segments are weighted with attention and reused to better model long-term dependencies, preventing **context fragmentation**.

The following approaches try to build on top of current structures to improve long-range modeling, reduce the parameter count, or optimize the computation performed by the model.

Compressive Memory

Introduced in: Compressive Transformers for Long-Range Sequence Modelling by Rae et al.

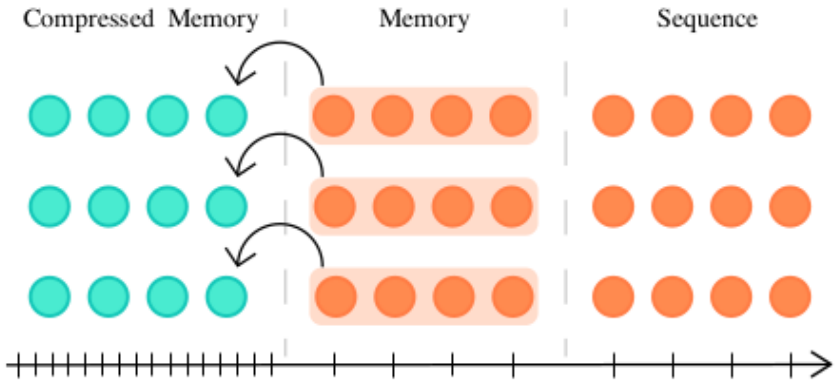


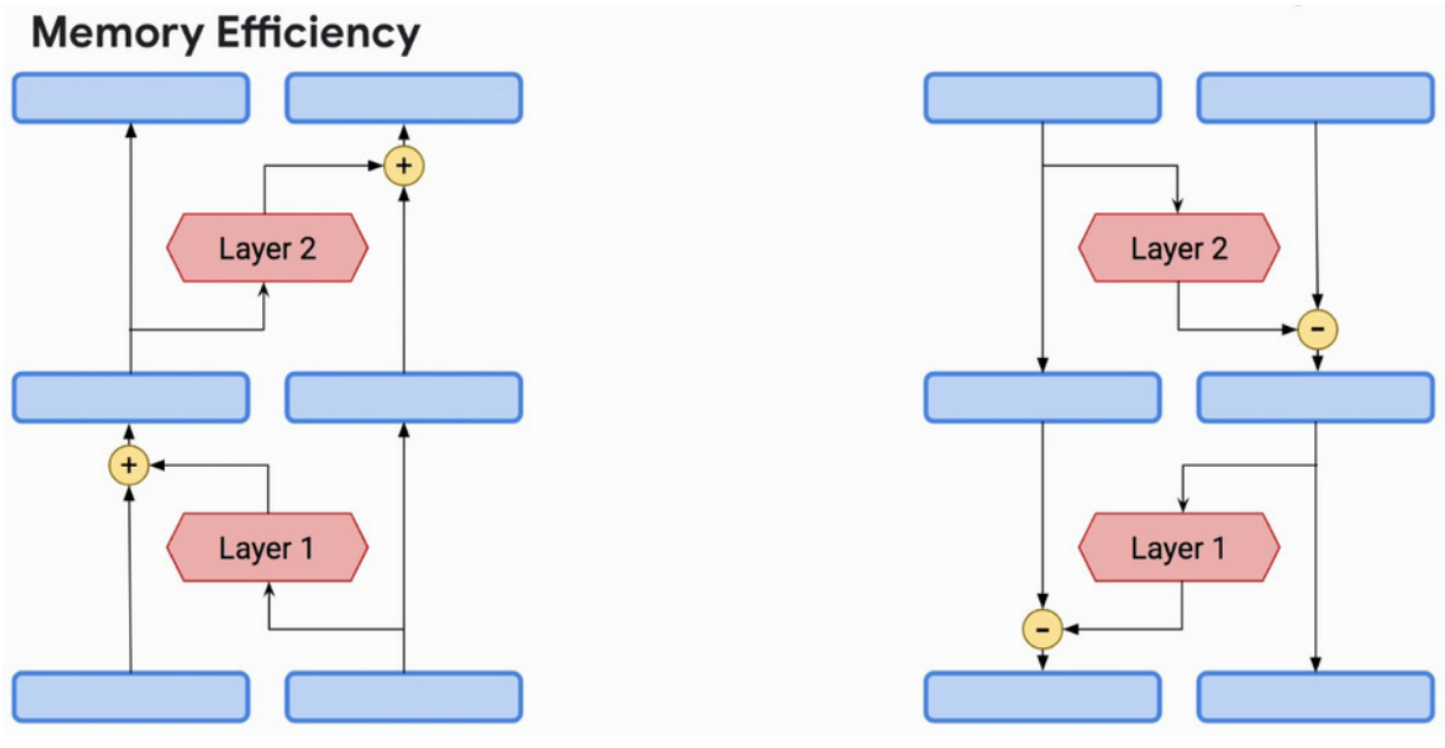
Figure 1: The Compressive Transformer keeps a fine-grained memory of past activations, which are then compressed into coarser representations that represent information from the distant past.

In Transformer-XL's recurrent memory approach, old memories are discarded to enable the storing of new ones in a first-in-first-out fashion. This method accounts only for recency, not taking into account the relevance of information that might get discarded.

Compressive Transformers builds upon the memory notion by adding a new **compressed memory** that stores coarse representations of older memories instead of discarding them. Authors try multiple alternatives for the compression function, finally selecting an **attention-reconstruction loss** that discards information that is not attended by the network. The use of compressive memory shows large improvements over the modeling of infrequent words, with empirical evidence of the network learning to preserve salient information through the compression mechanism.

Reversible Layers

Introduced in: Reformer: The Efficient Transformer by Kitaev et al.



The main idea behind **reversibility** is to enable the recovering of activations in any layer of the network by using only activations of the following layer and model parameters. This feature is especially interesting when applied to Transformer models since they are usually composed of a large pile of stacked layers and their memory complexity grows linearly with the layer count.

Reformer introduces reversibility in the Transformer architecture by combining attention and feed-forward sublayers into a single reversible layer. This allows to store activations only for the topmost layer and recover all the other ones by reversing layers during back-propagation, making the model depth irrelevant memory-wise. Further improvements in memory complexity are achieved by **chunking** independent computations in feed-forward and reversible layers.

Cross-Layer Parameter Sharing

Introduced in: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations by Lan et al.

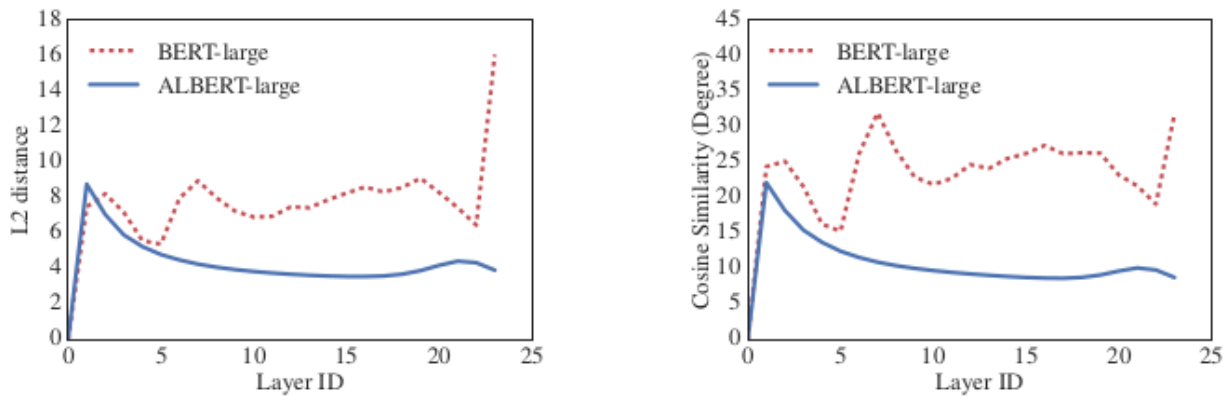


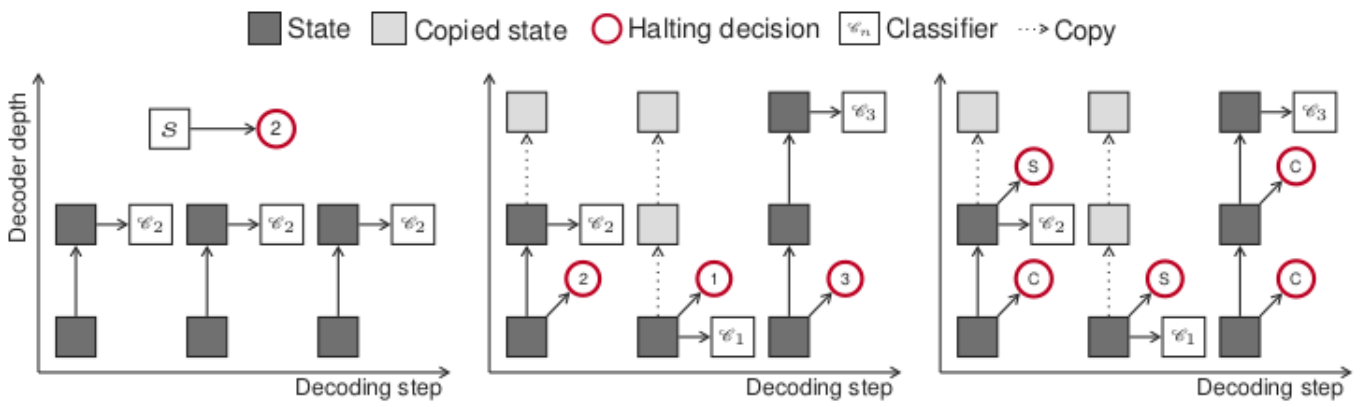
Figure 1: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

A simple yet very effective approach to greatly reduce the parameter count inside deep Transformer models is to share parameters across multiple layers, as it was shown in the Universal Transformer paper at ICLR 2019.

ALBERT authors experiment cross-layer parameter sharing for both self-attention and feed-forward sublayers, finding that sharing both weight matrices contributes to bringing down the total parameter count of the model by a factor of $7\times$ (for embedding size $E = 128$) while only slightly affecting final performances. The use of parameter sharing leads to smoother transition across layers and effectively stabilizes network parameters.

Adaptive Depth Estimation

Introduced in: Depth-Adaptive Transformer by Elbayad et al.



(a) Sequence-specific depth (b) Token-specific - Multinomial (c) Token-specific - Geometric-like

Figure 2: Variants of the adaptive depth prediction classifiers. Sequence-specific depth uses a multinomial classifier to choose an exit for the entire output sequence based on the encoder output s (2a). It then outputs a token at this depth with classifier $\mathscr{C}_n$. The token-specific multinomial classifier determines the exit after the first block and proceeds up to the predicted depth before outputting the next token (2b). The token geometric-like classifier (2c) makes a binary decision after every block to dictate whether to continue (C) to the next block or to stop (S) and emit an output distribution.

Current models perform a fixed number of computations for each input, regardless of the underlying complexity specific to each sequence. This problem was already highlighted in the Universal Transformer, which proposes a repeated application of the same layer with **adaptive computation time (ACT)**, but the resulting increase in per-layer weights considerably reduce the overall network speed.

Depth-adaptive Transformer solves this issue by encoding a sequence with a standard Transformer encoder and decoding it with a variable number of steps. To do so, a classifier is attached to each repeated layer of the decoder and the whole set is then trained with **aligned** and **mixed training** (see image) using the **anytime prediction** approach first introduced in the field of computer vision. Authors explore different mechanisms to adaptively control the amount of computation both on sequence level and on a per-token basis and conclude that an adaptive reduction of more than 75% of decoder layers can be applied without any loss in accuracy on machine translation tasks.

Conclusion

Many of the approaches introduced at ICLR 2020 offer widely applicable solutions to specific problems that characterize the original Transformer architecture, ranging from the self-attention computation to the model structure itself.

Many of these approaches seem promising for future developments of the Transformer and, most importantly, are likely to bring complementary improvements once many of them included in a single architecture.

My hope for ICLR 2021 is to see more incremental work that puts together already-existing strategies to highlight the most effective combinations between them.

See also: What's new for Transformers at the ICLR 2020 Conference? by Sergi Castella

Natural Language Processing Language Modeling Deep Learning Transformers ICLR2020 Word Embeddings Self-Attention

Related

- Neural Language Models: the New Frontier of Natural Language Understanding
- Covid-19 Semantic Browser
- Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks
- The Literary Ordnance: When the Writer is an AI
- Altalo Svevo: Letters from an Artificial Intelligence