Appendices

A Other ABC Models

A.1 Sparse Local-to-global Attention

It sparsifies attention pattern to reduce the number of tokens that are attended to (Beltagy et al., 2020; Zaheer et al., 2020, *inter alia*). All queries attend to a subset of $n < N$ "global tokens," while ignoring others. Therefore the effective context size is reduced to n. The global tokens are usually pre-selected by positions according to using some heuristics. Local-to-global attenion is an instance of ABC: it can be recovered by letting $\boldsymbol{\phi}_t = \mathbf{e}_i$ if x_t is the ith global token ($i = 1, \dots, n$), and the zero vectors for others.

A.2 Random Memory Control

As a baseline, ABC_{RD} stores each token in a randomly-selected memory slot. This is achieved by letting $\boldsymbol{\phi}_t = \mathbf{e}_{i_t}$, where i_t is uniformly drawn from $\{1, \dots, n\}$ for each t. It is designed as a baseline to ABC_{MLP} and Linformer to quantify the differences between random and learned bounded memory control.

Random sparse attention patterns are explored by Zaheer et al. (2020), where a subset of $n < N$ tokens are randomly selected to be attended to by all. ABC_{RD} is different, and it attends to *all* tokens, but "squash" them into an n-slot memory.

A.3 Compressive Transformer with Mean Pooling

The compressive transformer (Rae et al., 2020) explores various ways to "squash" long context into smaller and more compact representations. It achieves state-of-the-art performance on several language modeling benchmarks. We show that at least the mean-pooling variant of the compressive transformer can be seen as an ABC instance.

The mean-pooling variant of the compressive transformer compresses the context by

$$\begin{aligned}&\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_N]^\top \in \mathbb{R}^{N \times d} \\ \to &\widetilde{\mathbf{K}} = \big[\underbrace{(\mathbf{k}_1 + \cdots + \mathbf{k}_c)}_{c}/c, \underbrace{(\mathbf{k}_{c+1} + \cdots + \mathbf{k}_{2c})}_{c}/c \dots, \underbrace{(\mathbf{k}_{N-c+1} + \cdots + \mathbf{k}_N)}_{c}/c\big]^\top \in \mathbb{R}^{n \times d}.\end{aligned}$$

where $c = N/n$ is the compression ratio. Here $N \bmod n = 0$ is assumed, since otherwise the sequence can be padded to.

The above model is an ABC instance by letting

$$\boldsymbol{\phi}_i = \mathbf{e}_{\lfloor (i-1)/c \rfloor + 1}/c. \tag{14}$$

A.4 Dilated Convolution Attention Patterns

The dilated attention pattern is similar to the sliding window attention and only considers the context within a predefined window. It differs in that it attends to every other token:

$$\widetilde{\mathbf{K}}_t = [\mathbf{k}_{t-2n+2}, \mathbf{k}_{t-2n+4}, ..., \mathbf{k}_{t-2}, \mathbf{k}_t]^\top. \tag{15}$$

It can be simulated with two separate queues $\widetilde{\mathbf{K}}^{\text{odd}}$ and $\widetilde{\mathbf{K}}^{\text{even}}$:

$$\widetilde{\mathbf{K}}^{\text{odd}}_t = \begin{cases} \mathbf{U}\widetilde{\mathbf{K}}^{\text{odd}}_{t-1} + \mathbf{e}_n \otimes \mathbf{k}_t, & \text{if } t \text{ is odd} \\ \widetilde{\mathbf{K}}^{\text{odd}}_{t-1}, & \text{otherwise} \end{cases} \qquad \widetilde{\mathbf{K}}^{\text{even}}_t = \begin{cases} \mathbf{U}\widetilde{\mathbf{K}}^{\text{even}}_{t-1} + \mathbf{e}_n \otimes \mathbf{k}_t, & \text{if } t \text{ is even} \\ \widetilde{\mathbf{K}}^{\text{even}}_{t-1}, & \text{otherwise} \end{cases}$$

Likewise for the values. Depending on t, the query attends to one of the two queues:

$$\text{output} = \begin{cases} (\widetilde{\mathbf{V}}^{\text{odd}})^\top \text{softmax}(\widetilde{\mathbf{K}}^{\text{odd}}\mathbf{q}_t), & \text{if } t \text{ is odd} \\ (\widetilde{\mathbf{V}}^{\text{even}})^\top \text{softmax}(\widetilde{\mathbf{K}}^{\text{even}}\mathbf{q}_t), & \text{otherwise.} \end{cases}$$

The above implementation could incur considerable amount of overhead and may be actually more expensive than the the original dilated window formulation. Therefore it has more conceptual value than practical one.

A.5 Shared Workspace and Linear Unified Nested Attention

Concurrently to this work, shared workspace (SW; Goyal et al., 2021) and linear unified nested attention (LUNA; Ma et al., 2021) also learn contextualized memory control strategies. Both can be seen as instances of ABC. At layer ℓ, their $\boldsymbol{\phi}_i^\ell$ is a function of previous layer's memory $\widetilde{\mathbf{X}}^{\ell-1} \in \mathbb{R}^{n\times d}$ and current layer's input $\mathbf{X}^\ell \in \mathbb{R}^{N\times d}$:

$$\boldsymbol{\phi}_i = \left[\mathrm{softmax}\left(\widetilde{\mathbf{X}}^{\ell-1}\mathbf{X}^{\ell\top}\right)\right]_{:,i}, \tag{16}$$

where $[\cdot]_{:,i}$ denotes the ith column of a matrix. Query, key, and value projections are suppressed for notation clarity.

SW and LUNA reveal the entire sequence to the control vectors, by constructing $\boldsymbol{\phi}$ as a function of previous layer's memory. Although both admit the recurrent computation as all ABC models do, they are ill-suited for causal attention and autoregressive decoding, since future information is "leaked" to $\boldsymbol{\phi}_i$ from the previous layer. LUNA resorts to a variant of Katharopoulos et al. (2020) in causal attention (Ma et al., 2021). In contrast, ABC_{MLP} never depends $\boldsymbol{\phi}_i$ on previous layer's memory, but only on current layer's input.

B More Details about ABC-MLP

B.1 Normalization in Causal Attention

An equivalent implementation to Eq. 12 is to normalize $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$ instead of $\boldsymbol{\phi}_i$ vectors:

$$\boldsymbol{\alpha}_i = \mathbf{exp}\left(\mathbf{W}_{\boldsymbol{\phi}}\mathbf{x}_i\right), \quad \boldsymbol{\phi}_i = \boldsymbol{\alpha}_i,$$

$$\bar{\mathbf{K}} = \widetilde{\mathbf{K}} \Big/ \sum_{j=1}^N \boldsymbol{\alpha}_j. \quad \bar{\mathbf{V}} = \widetilde{\mathbf{V}} \Big/ \sum_{j=1}^N \boldsymbol{\alpha}_j.$$

$$\text{output} = \bar{\mathbf{V}}^\top \mathrm{softmax}(\bar{\mathbf{K}}\mathbf{q}).$$

$\mathbf{M}/\mathbf{z}$ divides the ℓth row of matrix $\mathbf{M}$ by vector $\mathbf{z}$'s ℓth dimension. This admits a linear complexity computation graph for the causal variant of ABC_{MLP}.

B.2 Higher-dimensional Case of Example 1

This section generalizes Example 1 to higher dimensional cases. Assume that the constant-sized memory has n slots. $\boldsymbol{\phi}_i$ is cauculated as in Eq. 12. Then $\widetilde{\mathbf{K}} = \sum_{i=1}^N \boldsymbol{\phi}_i \otimes \mathbf{k}_i \in \mathbb{R}^{n\times d}$. Each row of $\widetilde{\mathbf{K}}$ can be seen as a separate attention mechanism with a pseudo query. Let $[\cdot]_\ell$ denote the ℓth row/dimension of a matrix/vector. Then for any $\ell = 1, \ldots, n$

$$\left[\widetilde{\mathbf{K}}\right]_\ell = \sum_{i=1}^N [\boldsymbol{\phi}_i]_\ell \otimes \mathbf{k}_i = \sum_{i=1}^N \frac{\exp([\mathbf{W}_\phi]_\ell \cdot \mathbf{x}_i)}{\sum_{j=1}^N \exp([\mathbf{W}_\phi]_\ell \cdot \mathbf{x}_j)} \mathbf{k}_i^\top$$

$$= \mathrm{attn}\left([\mathbf{W}_\phi]_\ell, \{\mathbf{x}_i\}_{i=1}^N, \{\mathbf{k}_i\}_{i=1}^N\right)^\top \in \mathbb{R}^{1\times d}.$$

In other words, there are n attention mechanisms in total, each with a separately-parameterized pseudo-query $[\mathbf{W}_\phi]_\ell$. They summarize the context for n times in parallel, each producing a d-dimensional vectors. These output vectors are then stacked into n-by-d memory $\widetilde{\mathbf{K}}$. $\widetilde{\mathbf{V}}$ is likewise.

B.3 Complexity Analysis for ABC

Table 6 compares ABC's time and space complexity against softmax attention's. Its savings come from a per-query complexity reduction from $\mathcal{O}(N)$ to $\mathcal{O}(n)$, for both time and space. Since the memory $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$ is shared across different queries, ABC comes with a linear complexity in sequence lengths. Therefore significant efficiency improvements can be achieved when $n \ll N$.

	Time Complexity			Space Complexity		
Model	**Mem.**	**Per Query**	**Overall**	**Mem.**	**Per Query**	**Overall**
Softmax Attn.	-	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$	-	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$
ABC	$\mathcal{O}(N)$	$\mathcal{O}(n)$	$\mathcal{O}(nN)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(nN)$

Table 6: ABC's time and space complexity in sequence length against the softmax attention's. "Mem." indicates the time and space needed for calculating and storing memory $\widetilde{\mathbf{K}}, \widetilde{\mathbf{V}}$. N denotes the sequence length, and n the memory size. The time complexity analysis assumes that the softmax attention *cannot* be parallelized across the queries. In practice, this is common in, e.g., autoregressive decoding, or for long sequences where the accelerators (e.g., GPUs) do not have enough threads to fully parallelize softmax attention's computation across different queries.

The same analysis applies to ABC_{MLP}.

C Experimental Details

C.1 Language Modeling

We closely build on Baevski & Auli (2019) and Kasai et al. (2021b). The hyperparameters are summarized in Table 9. All models are trained on 4 A100 GPUs.

C.2 Machine Translation

We experiment with a sentence-level (WMT14 EN-DE, Bojar et al., 2014) and a document-level benchmark (IWSLT14 ES-EN, Cettolo et al., 2014) to assess model performance over various sequence lengths. The preprocessing and data splits of WMT14 EN-DE follow Vaswani et al. (2017). A 32,768 byte pair encoding (BPE; Sennrich et al., 2016) vocabulary is shared between source and target languages. For IWSLT14, we follow Miculicich et al. (2018) and use the *dev2010* subset for development and *tst2010-2012* for testing. The tokenization is also the same as Miculicich et al. (2018): we tokenize and truecase Spanish and English with Moses (Koehn et al., 2007) and run byte-pair encoding with 30k splits, shared between the two languages. The final dataset contains 1421, 8, and 42 documents for training, development, and testing. On average, each document contains 126.7 sentences and each sentence contains 21.7(ES)/22.5(EN) BPE. We use a sliding window with length-4 and stride-one to generate our dataset. During inference, we use predicted context at the target side.

We average the checkpoints from the last five epochs to obtain the final model Vaswani et al. (2017). In inference, we apply beam search with size 5 and length penalty 0.6. Other hyperparameters are summarized in Table 10. All models are trained on 4 RTX 2080 Ti GPUs.

Additional machine translation results. In addition to the results presented in §5.2, Table 7 further compares, on the WMT14 EN-DE dataset, the clustering-based (§3.2) and sliding-window (§3.3) models of ABC, as well as ReLU and $\mathrm{sigmoid}$ variants of ABC_{MLP}. Clustering and sliding-window ABC variants *underperform* ABC_{MLP} with the same memory sizes by more than 0.5 BLEU. Both ReLU and $\mathrm{sigmoid}$ underperform their $\exp$ counterpart. We observe that ABC_{MLP}-ReLU suffers a severe "the rich gets richer" issue: all tokens are stored in a handful of slots, no matter how large the memory size is. This could be the reason for its suboptimal accuracy. Further investigations are deferred to future work.

MLP-$\exp$-all replaces the encoder's softmax attention modules with ABC, in addition to the decoder's. It underperforms ABC_{MLP} by 0.3 BLEU.

Figure 2 compares ABC_{MLP}'s (32-8 memory sizes) attention memory overhead with softmax attention's. Following Kasai et al. (2021b), we consider a synthetic sequence-to-sequence generation task with varying sequence lengths. A batch size of 16 and greedy decoding is used. The models are of the same size as those in §5.2.

Model	ϕ	Cross n	Causal n	Encoder n	BLEU
BASE	-	-	-	-	27.2
ABC	Window	32	32	-	26.3
	Cluster	32	32	-	26.8
	MLP-ReLU	32	8	-	-
	MLP-ReLU	32	32	-	26.4
	MLP-sigmoid	32	8	-	26.8
	MLP-sigmoid	32	32	-	27.0
	MLP-exp	32	8	-	27.1
	MLP-exp	32	32	-	**27.3**
	MLP-exp-all	32	32	32	27.0

Table 7: ABC variants' SacreBLEU on WMT14 EN-DE sentence-level machine translation test set. MLP-ReLU with 32/8 memory sizes fails to converge. MLP-exp-all applies ABC in both the encoder and the decoder, while others only in the decoders.

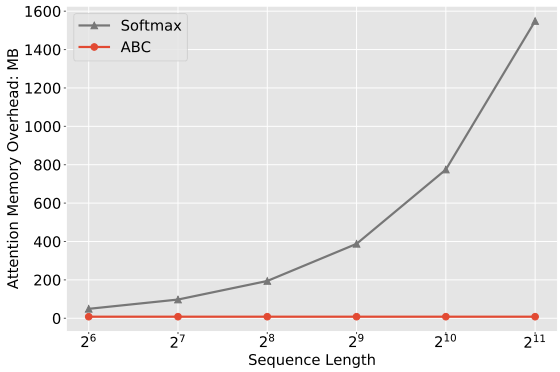

Figure 2: Machine translation decoding memory overhead of the attention computation. The setting follows Kasai et al. (2021b), with greedy decoding and batch size 16.

C.3 MASKED LANGUAGE MODEL FINETUNING

Our data for continued pretraining is a concatenation of BookCorpus (Zhu et al., 2015), English Wikipedia, OpenWebText (Gokaslan & Cohen, 2019), and RealNews (Zellers et al., 2019). Our data differs from RoBERTa's pretraining data, which we do *not* have access to. We replace their CC-News (Nagel, 2016) with RealNews, and drop Stories (Trinh & Le, 2018). At the time of this project, the public access to the Stories dataset is broken.[9] Our machine does *not* have a large enough memory to load all the data. We therefore split the training data into 20 shards, after shuffling. Other preprocessing is the same as Liu et al. (2019).[10] The hyperparameters for continued pretraining follows base-sized RoBERTa, part of which summarized in Table 11. All models are trained on a single TPU v3 accelerator.

For downstream task finetuning, we use the same hyperparameters as Liu et al. (2019).[11] Table 12 briefly introduces the tasks. The readers are referred to Wang et al. (2018) for futher details.

[9] https://console.cloud.google.com/storage/browser/commonsense-reasoning/reproduce/stories_corpus?pli=1

[10] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.pretraining.md

[11] https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.glue.md

Data	Train	Dev.	Test	Vocab.	Sent./doc
WikiText-103	103M	218K	246K	268K	-
WMT14 EN-DE	4.5M	3K	3K	32K	-
IWSLT14 ES-EN	1713	8	56	30K	121.5

Table 8: Statistics for the datasets. WikiText-103 split sizes are in number of tokens, WMT14 in number of sentences, and IWSLT14 in number of documents.

Hyperprams.	Baevski & Auli (2019)	Kasai et al. (2021b)
# Layers	16	32
# Heads	8	8
Embedding Size	1024	1024
Head Size	128	128
FFN Size	4096	4096
Batch Size	64	64
Learning Rate	1.0	1.0
Dropout	0.3	0.3
Layer Dropout	-	0.2
Memory size	$[32, 64]$	64

Table 9: Hyperparameters used in the language modeling experiments.

Hyperprams.	WMT14	IWSLT14
# Layers	6	6
# Heads	8	8
Embedding Size	512	512
Head Size	64	64
FFN Size	2048	1024
Warmup Steps	6000	4000
Dropout	0.1	0.3
Cross Attention Memory Size	32	128
Causal Attention Memory Size	8	64

Table 10: Hyperparameters used in the machine translation experiments.

Hyperprams.	Values
# Layers	12
# Heads	12
Embedding Size	768
Head Size	64
FFN Size	3072
Dropout	0.1
Memory Size	$[64, 128]$

Table 11: Hyperparameters for continued pretraining in the masked language model finetuning experiments.

Data	Task	Train	Dev.
MNLI	Entailment	392K	9.8K
QNLI	Entailment	105K	5.5K
QQP	Paraphrase	363K	40K
SST-2	Sentiment	67K	873

Table 12: GLUE datasets and statistics. MNLI: Williams et al. (2018); QNLI is compiled by GLUE's authors using Rajpurkar et al. (2016); QQP: Csernai (2017, accessed September 1, 2020); SST-2: Socher et al. (2013).