Compressive Transformers For Long-Range Sequence Modelling

ICLR 2020

Citation count: 8

Jack W. Rae*† ‡ Anna Potapenko*† Siddhant M. Jayakumar† Chloe Hillier†

Timothy P. Lillicrap†‡

Motivation

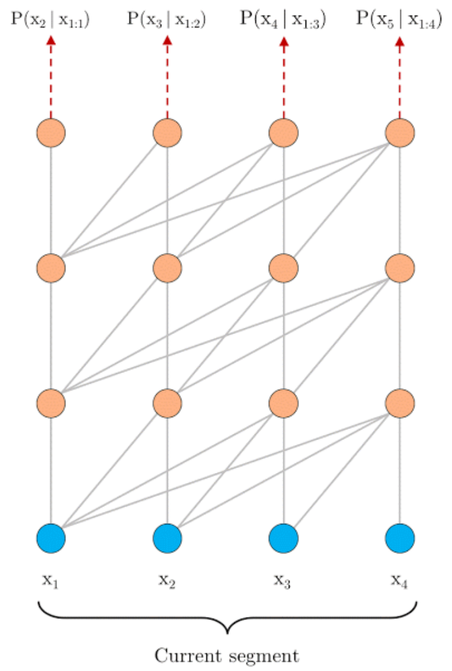
- Transformers need large memory to store previous experience and compute the longer attention.
- Hard to approach **long and sparse attention**
- How about we **compress the old memory and only retain the most important information**?

Background

- Transformer XL (Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context)
- Segment-Level Recurrence
- Relative Positional Encodings

Background

- Segment-Level Recurrence



Every hidden layer caches the previous state of itself and concatenate to the previous segment.

Background

- Segment-Level Recurrence

$$\mathbf{s}_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}]$$

$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

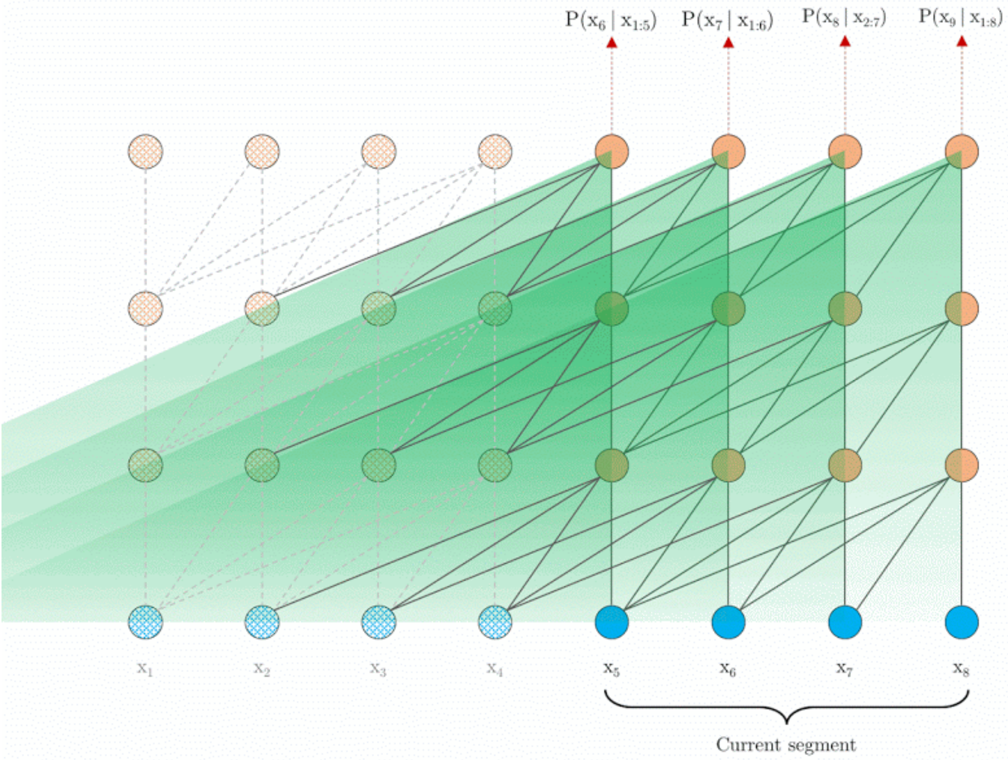
$\mathbf{s}_{\tau+1}$: Consecutive segments of length L

$\mathbf{h}_{\tau}^{n} \in \mathbb{R}^{L\times d}$: n-th Hidden layer for τ –th segment, d is the hidden dimension

SG() : Function for Stop Gradient

$[\mathbf{h}_u \circ \mathbf{h}_v]$: The operator denotes the concatenation of two hidden segments

Background



P(x6 | x1:5)
P(x7 | x1:6)
P(x8 | x2:7)
P(x9 | x1:8)
x1
x2
x3
x4
x5
x6
x7
x8
Current segment

Problem formulation

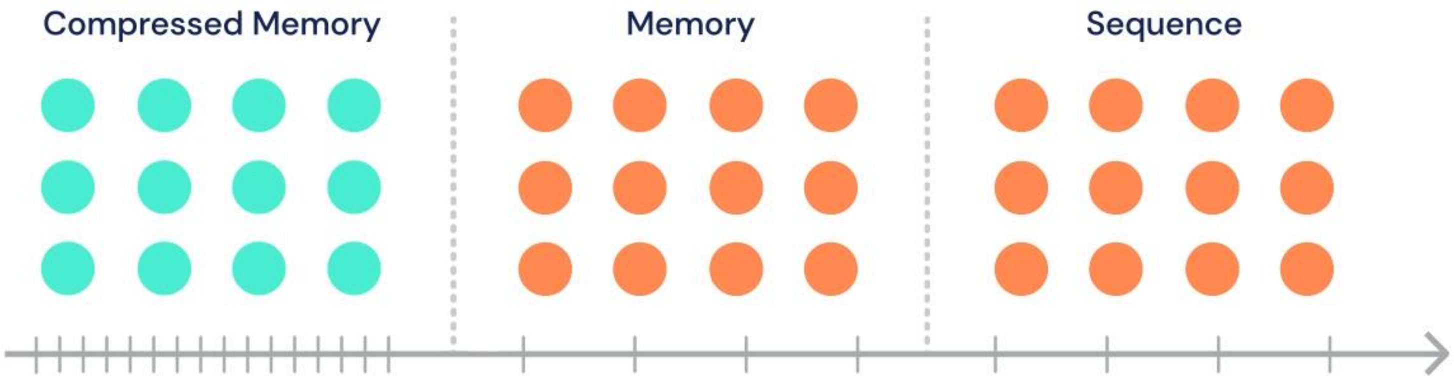
- Is there a cheaper way to increase the attention size?

Outline

- Main Idea
- Experiments & Evaluation
- Conclusion

Main Idea

- **Compressive Memory**
- Based on transformer XL, Compress old memories, and store them in an additional compressed memory. In this example compression ratio C = 3



Main Idea

Algorithm 1 Compressive Transformer

At time zero

$\mathbf{m_0} \leftarrow \mathbf{0}$ // Initialize memory to zeros $(l \times n_m \times d)$

$\mathbf{cm_0} \leftarrow \mathbf{0}$ // Initialize compressed memory to zeros $(l \times n_{cm} \times d)$

At time t

$\mathbf{h}^{(1)} \leftarrow \mathbf{xW_{emb}}$ // Embed input sequence$(n_s \times d)$

for layer $i = 1, 2, \ldots, l$ **do**

$\quad \mathbf{mem^{(i)}} \leftarrow \text{concat}(\mathbf{cm_t^{(i)}}, \mathbf{m_t^{(i)}})$ // $((n_{cm} + n_m) \times d)$

$\quad \tilde{\mathbf{a}}^{(\mathbf{i})} \leftarrow \text{multihead\_attention}^{(i)}(\mathbf{h^{(i)}}, \mathbf{mem_t^{(i)}})$ // MHA over both mem types $(n_s \times d)$

$\quad \mathbf{a^{(i)}} \leftarrow \text{layer\_norm}(\tilde{\mathbf{a}}^{(\mathbf{i})} + \mathbf{h^{(i)}})$ // Regular skip + layernorm $(n_{cm} \times d)$

$\quad \mathbf{old\_mem^{(i)}} \leftarrow \mathbf{m_t^{(i)}}[: n_s]$ // Oldest memories to be forgotten $(n_s \times d)$

$\quad \mathbf{new\_cm^{(i)}} \leftarrow f_c^{(i)}(\mathbf{old\_mem^{(i)}})$ // Compress oldest memories by factor c $(\lfloor \frac{n_s}{c} \rfloor \times d)$

$\quad \mathbf{m_{t+1}^{(i)}} \leftarrow \text{concat}(\mathbf{m_t^{(i)}}, \mathbf{h^{(i)}})[-n_m :]$ // Update memory $(n_m \times d)$

$\quad \mathbf{cm_t^{(i)}} \leftarrow \text{concat}(\mathbf{cm_t^{(i)}}, \mathbf{new\_cm^{(i)}})[-n_{cm} :]$ // Update compressed memory $(n_{cm} \times d)$

$\quad \mathbf{h^{(i+1)}} \leftarrow \text{layer\_norm}(\text{mlp}^{(i)}(\mathbf{a^{(i)}}) + \mathbf{a^{(i)}})$ // Mixing MLP $(n_s \times d)$

cm: Compressive Memory

f_c: Compression Operator

Main Idea

For choices of **Compression Functions fc**

- **Max/Mean Pooling:** where the kernel and stride is set to the compression rate c
- **1D convolution:** With kernel & stride set to c, need to be optimized
- **Dilated Convolutions:** need to be optimized
- **Most-Used:** where the memories are sorted by their average attention (usage) and the most-used are preserved.

Main Idea

For choices of **Compression Loss**

- **BPTT:** Backpropagating through time over unroll, but cost time
- **Auto-Encoding Loss:** Reconstruct old memory from compressed memory, attempt to retain all information

$$\mathcal{L}^{ae} = ||\mathbf{old\_mem^{(i)}} - g(\mathbf{new\_cm^{(i)}})||_2$$

- **Attention Reconstruction**: Next Page

Main Idea

Attention Reconstruction

Algorithm 2 Attention-Reconstruction Loss

$L^{attn} \leftarrow 0$
for layer $i = 1, 2, \ldots, l$ **do**
  $\mathbf{h^{(i)}} \leftarrow$ stop_gradient($\mathbf{h^{(i)}}$) // Stop compression grads from passing...
  $\mathbf{old\_mem^{(i)}} \leftarrow$ stop_gradient($\mathbf{old\_mem^{(i)}}$) // ...into transformer network.
  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow$ stop_gradient(attention params at layer i) // Re-use attention weight matrices.
  def attn$(\mathbf{h}, \mathbf{m}) \leftarrow \sigma((\mathbf{hQ})\,(\mathbf{mK}))(\mathbf{mV})$ // Use content-based attention (no relative).
  $\mathbf{new\_cm^{(i)}} \leftarrow f_c^{(i)}(\mathbf{old\_mem^{(i)}})$ // Compression network (to be optimized).
  $L^{attn} \leftarrow L^{attn} + ||\text{attn}(\mathbf{h^{(i)}}, \mathbf{old\_mem^{(i)}}) - \text{attn}(\mathbf{h^{(i)}}, \mathbf{new\_cm^{(i)}})||_2$

And we found that **Attention Reconstruction** works **Best** through 3 choice.

Main Idea

Book-Level language modelling benchmark: **PG-19**

- Release a new dataset: PG-19
- Using text from books published over 100 years
- Contains 28752 books, 11GB

	Avg. length (words)	Train Size	Vocab	Type
1B Word	27	4.15GB	793K	News (sentences)
Penn Treebank	355	5.1MB	10K	News (articles)
WikiText-103	3.6K	515MB	267K	Wikipedia (articles)
PG-19	69K	10.9GB	(open)	Books

Experiments & Evaluation

- ENWIK8
- PG-19
- WIKITEXT-103
- Compressibility Of Layers
- Attention
- Optimization Schedule
- Speech
- Reinforcement Learning

Experiment: PG-19

Table 3: Eval. perplexities on PG-19.

	Valid.	Test
36L TransformerXL	45.5	36.3
36L Compressive Transf.	$\boxed{43.4}$	$\boxed{33.6}$

During Training:
Transformer XL: 36 layers, window size 512, attention window 1024,
Compressive Transformer: 36 layers, window size 512, memory size 512, compressive memory 512, C = 2

About improve 7.4%

Experiment: Compression approaches on Enwik8

Compression fn	Compression loss	BPC
Conv	BPTT	0.996
Max Pooling	N/A	0.986
Conv	Auto-encoding	0.984
Mean Pooling	N/A	0.982
Most-used	N/A	0.980
Dilated conv	Attention	0.977
Conv	Attention	**0.973**

Sweep over compression rate of 2, 3, 4

Attention Reconstruction works best

Experiment: WIKITEXT-103

Table 6: Validation and test perplexities on WikiText-103.

	Valid.	Test
LSTM (Graves et al., 2014)	-	48.7
Temporal CNN (Bai et al., 2018a)	-	45.2
GCNN-14 (Dauphin et al., 2016)	-	37.2
Quasi-RNN Bradbury et al. (2016)	32	33
RMC (Santoro et al., 2018)	30.8	31.9
LSTM+Hebb. (Rae et al., 2018)	29.0	29.2
Transformer (Baevski and Auli, 2019)	-	18.7
18L TransformerXL, M=384 (Dai et al., 2019)	-	18.3
18L TransformerXL, M=1024 (ours)	-	18.1
18L Compressive Transformer, M=1024	**16.0**	**17.1**

During Training:
Compressive Transformer: memory size 500, compressive memory 1500, C = 4

5% higher probability on correct word than the prior SotA Transformer XL.

Experiment: WIKITEXT-103

	> 10K	1K–10K	100 – 1K	< 100	All
LSTM*	12.1	219	1,197	9,725	36.4
TransformerXL (ours)	7.8	61.2	188	1,123	18.1
Compressive Transformer	**7.6**	**55.9**	**158**	**937**	**17.1**
Relative gain over TXL	2.6%	9.5%	21%	19.9%	5.8%

During Training:

Compressive Transformer: memory size 500, compressive memory 1500, C = 4

Obtains a much larger **improvement of ≈ 20% for infrequent words**

Experiment: ENWIK8

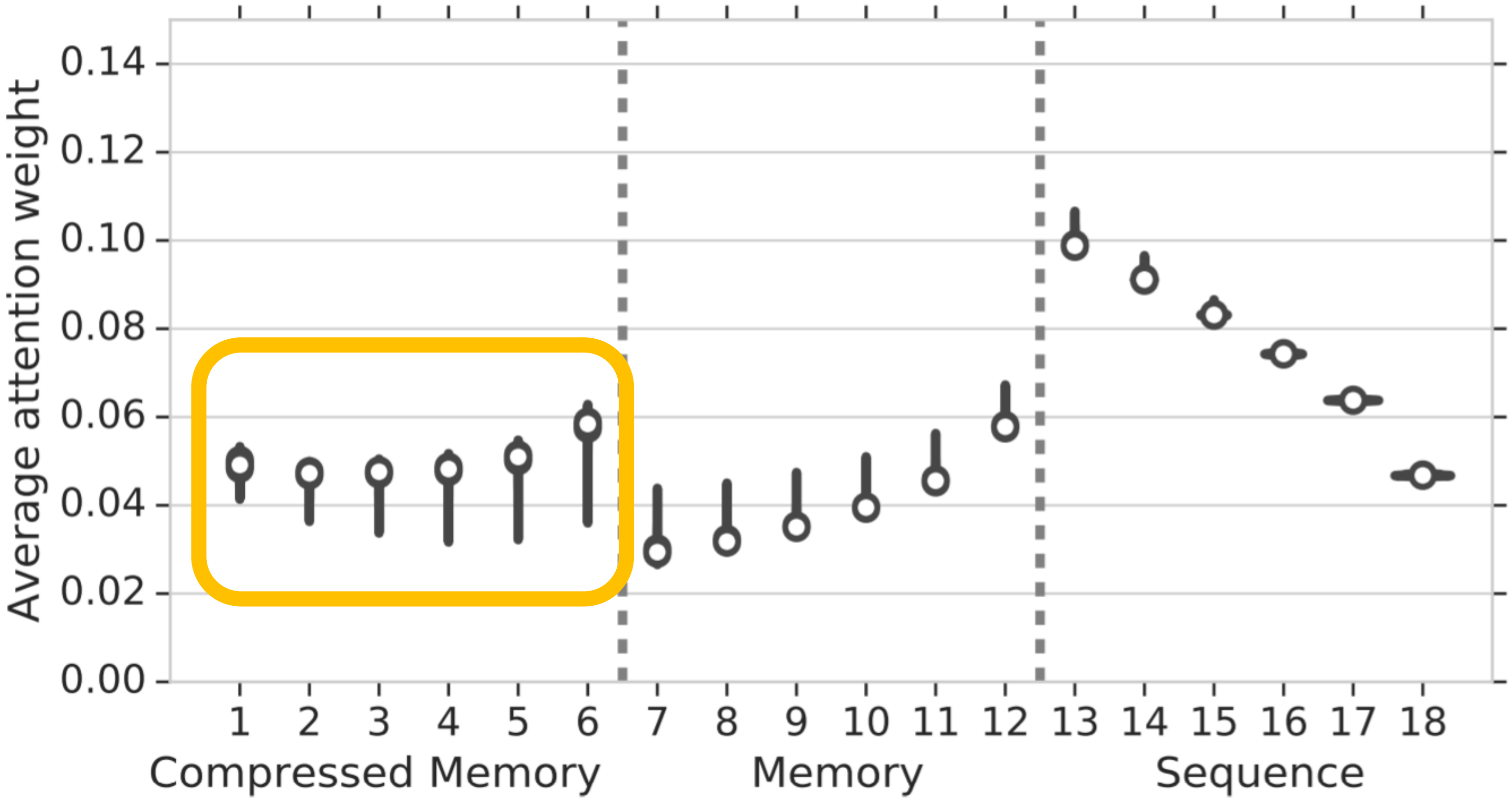
Table 4: State-of-the-art results on Enwik8.

Model	BPC
7L LSTM (Graves, 2013)	1.67
LN HyperNetworks Ha et al. (2016)	1.34
LN HM-LSTM Chung et al. (2016)	1.32
ByteNet (Kalchbrenner et al., 2016)	1.31
RHN Zilly et al. (2017)	1.27
mLSTM Krause et al. (2016)	1.24
64L Transf. Al-Rfou et al. (2019)	1.06
24L TXL (Dai et al., 2019)	0.99
Sparse Transf. (Child et al., 2019)	0.991
Adaptive Transf. (Sukhbaatar et al., 2019)	0.98
24L TXL (ours)	0.98
24L Compressive Transformer	**0.97**

During Training:
Transformer XL: 24 layers, window size 768, memory size 2304,
Compressive Transformer: 24 layers, window size 768, memory size 768, compressive memory 1152, C = 3

During Evaluation:
Transformer XL: memory size 4096
Compressive Transformer: memory size , compressive memory size 3072

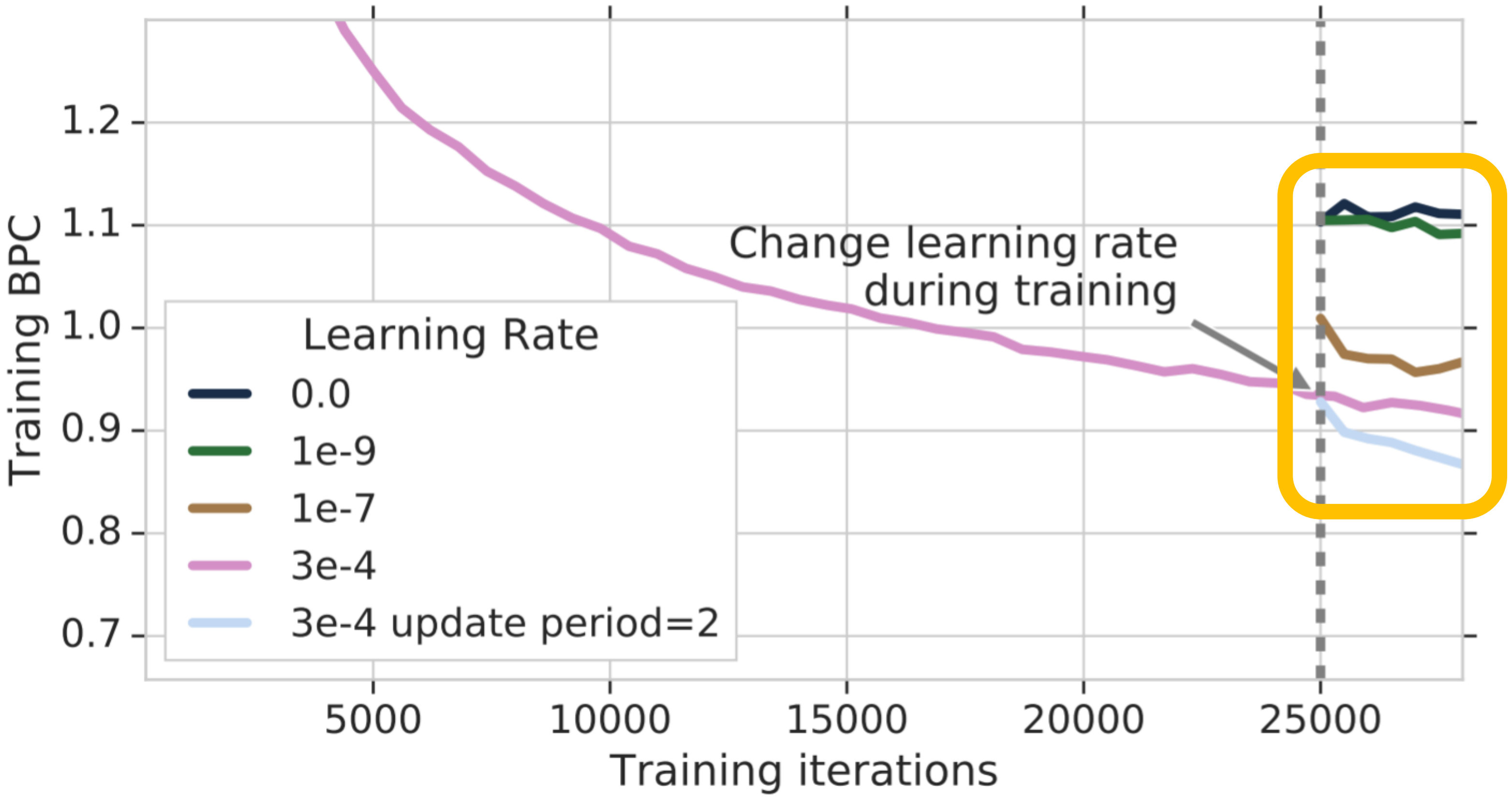
Experiment: Attention



It average the attention weight over a sample of 20, 000 sequences from a trained model on Enwik8 and separate the attention into eighteen buckets

It shows an **increase in the activations** stored in compressed memory.

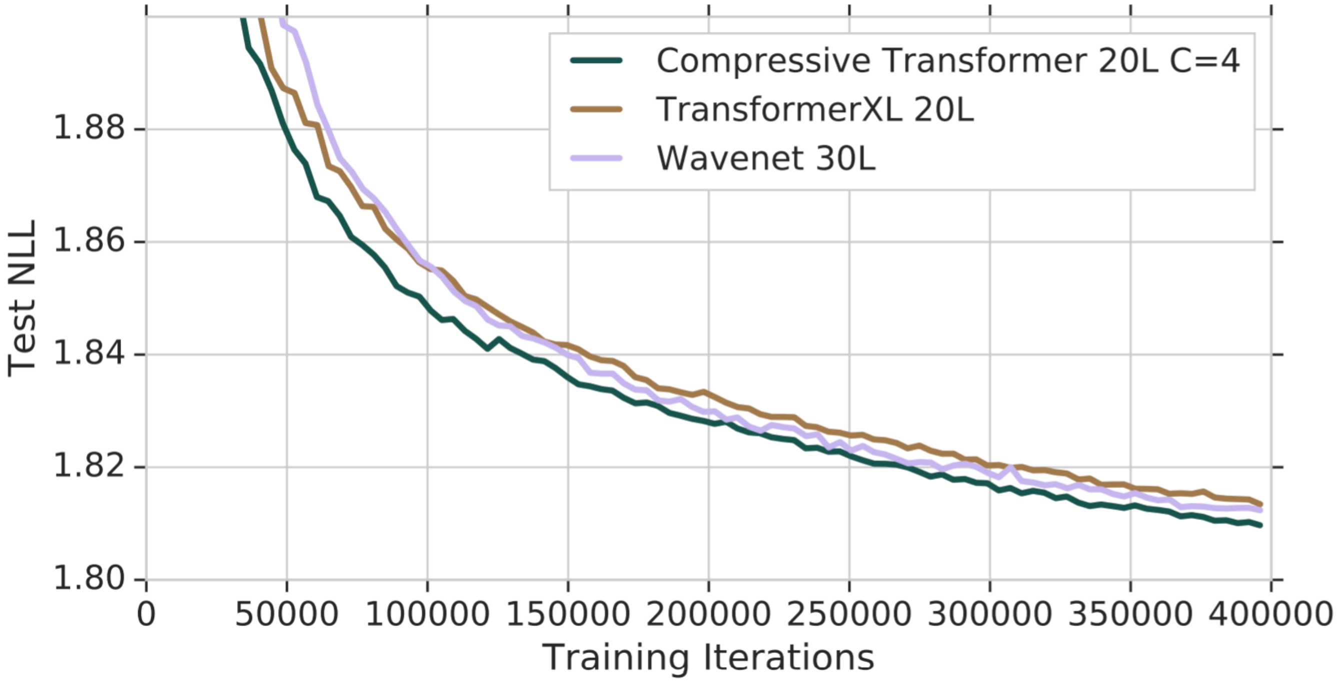
Experiment: Optimization Schedule



An observation that when the **learning rate is tuned to be much smaller, performance degrades drastically**

propose reducing the frequency of optimization updates during training(increases the effective batch size)

Experiment: Speech

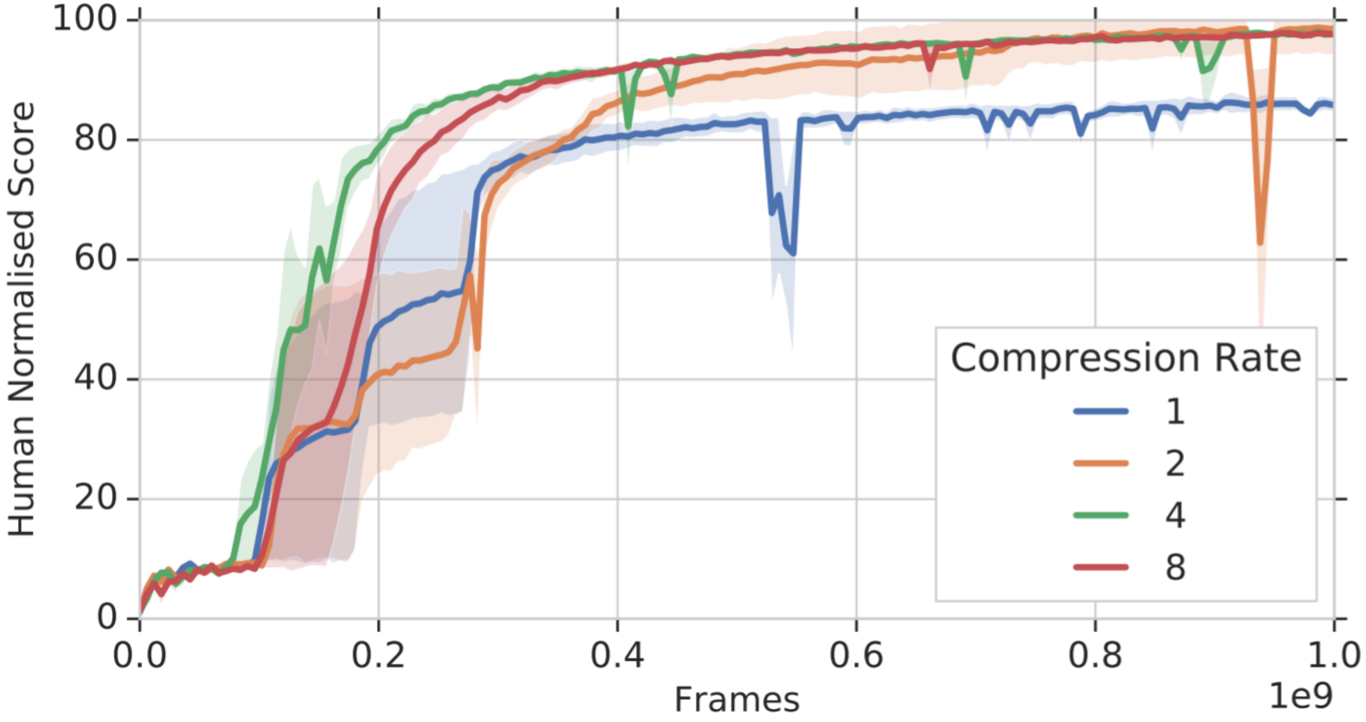


Compressive transformer is **Better** than WaveNet.

Train the model on 24.6 hours of 24kHz North American speech data.
Chunk the sequences into windows of size 3840, roughly 80ms of audio

During Training:
Transformer XL: 20 layers, window size 768, memory size 1568
Compressive Transformer: 20 layers, window size 768, memory size 768, compressive memory 768, C = 4
(But not full convergence, should be better)
WaveNet: 30 layers

Experiment: Reinforcement Learning



Replace an LSTM to Compressive Transformer in the IMPALA.

Task: DMLab-30 rooms select nonmatching object

Compressive transformer is able to solve the task to **Human-Level** but the model with **compression rate 1 is unable**

Conclusion

- Compression is a simpler approach to dynamic or sparse attention — which often requires custom kernels to make efficient.