Longformer: The Long-Document Transformer

Iz Beltagy, Matt Peters, Arman Cohan

Long documents:

- Many NLP tasks require processing full documents
  - e.g: QA, summarization, document-classification
- Semantic scholar: 90% percentile paper length = 7,000 tokens

Transformers: SOTA

- LSTM scales to long documents but doesn’t work as well

Long documents + Transformers is hard

- Self-attention is expensive: O(n^2)

Prior work

- Avoid working with long documents
  - e.g. MRQA: skip the question if the answer is not in the first 800 tokens

- Chunk, extract, combine
  - task and dataset specific
  - Loses global context

Prior work

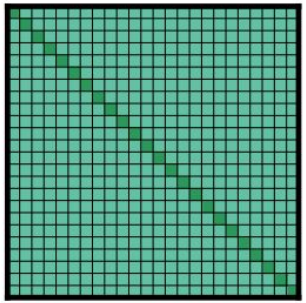
Model	attention matrix	char-lm	other tasks	pretrain
Transformer-XL (2019)	ltr	yes	no	no
Adaptive Span (2019)	ltr	yes	no	no
Compressive (2020)	ltr	yes	no	no
Reformer (2020)	sparse	yes	no	no
Sparse (2019)	sparse	yes	no	no
BP-Transformer (2019)	sparse	yes	MT	no
Blockwise (2019)	sparse	no	QA	yes
Our Longformer	sparse	yes	multiple	yes

Longformer - sparse attention matrix

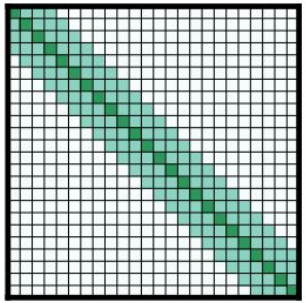
Avoid computing the full attention matrix

Tokens attend to each others following an “attention pattern”

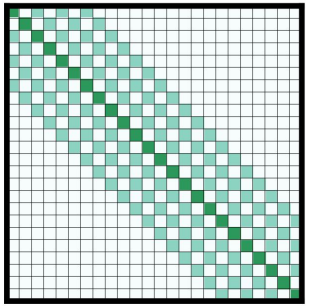
Large receptive field with stacked layers



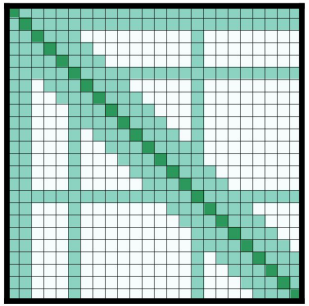
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window

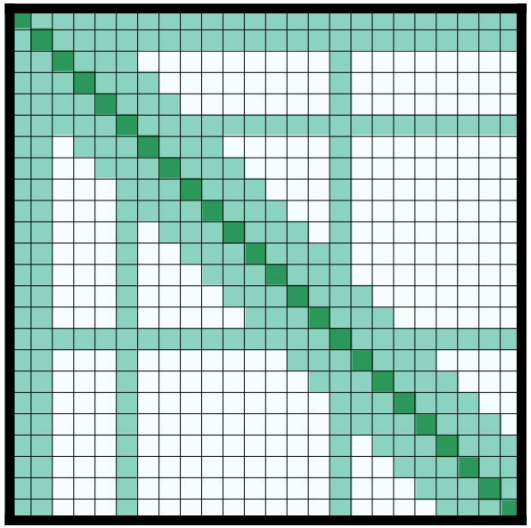


(d) Global+sliding window

Longformer - local and global attention

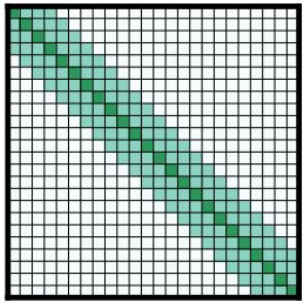
- Global attention is user defined based on the task
- Local attention - good token representation
- Global attention - flexibility to learn tasks
  - LM: local representation
  - Classification: aggregate seq into CLS
  - QA: compare context and question tokens
- Notice: indirect information flow is not enough.

  Direct attention is needed.



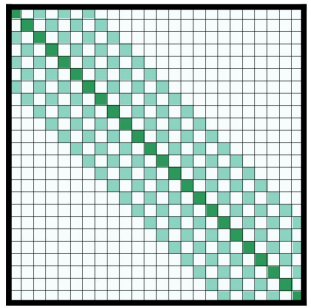
Implementation

- Required banded-multiplication is not supported in existing DL libraries
- Implementation 1: sliding chunks
  - Native PyTorch (easy to deploy and use)
  - Limited to the non-dilated case
  - Splits the sequence into chunks of size W and overlapping of size W/2, matrix multiply each chunk, then mask out the extra elements
  - Computed 2x more values than needed



Implementation

- Required banded-multiplication is not supported in existing DL libraries
- Implementation 2: custom cuda kernel
  - Implemented in TVM
    - Compiles python code into cuda c++
    - Easier to use than writing cuda from scratch
  - Supports dilation
  - Only computes the non-zero elements (memory efficient)
  - A bit harder to deploy and use



Performance

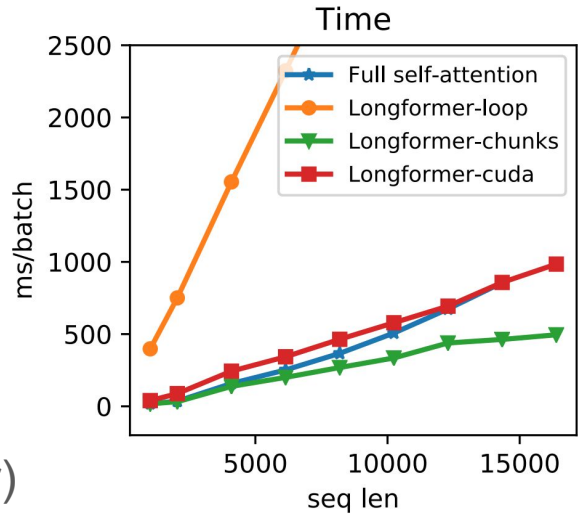
`loop` is a naive implementation using loops

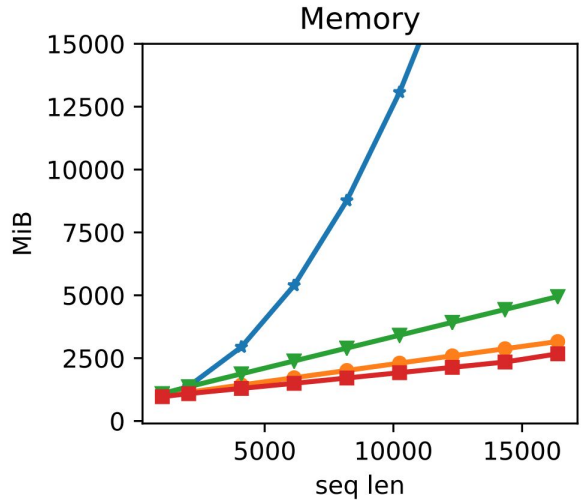
`loop` and `cuda` are the most memory efficient

`chunks` uses 2x more memory than `cuda` (but still linear)

`chunks` faster than `cuda` because it uses nvidia MM

Use `chunks` for pretrain/finetune, use `cuda` for char-lm





```python
import torch
import tvm
from tvm.contrib import dlpack

d1 = tvm.var('d1')  # define dimensions as variables
d2 = tvm.var('d2')  # define dimensions as variables
d3 = tvm.var('d3')  # define dimensions as variables
A = tvm.placeholder((d1, d2), name='A', dtype='float32')  # first tensor
B = tvm.placeholder((d2, d3), name='B', dtype='float32')  # second tensor
k = tvm.reduce_axis((0, d2), name='k')  # dimension to sum over
output_shape = (d1, d3)
algorithm = lambda i, j: tvm.sum(A[i, k] * B[k, j], axis=k)  # explain computation
R = tvm.compute(output_shape, algorithm, name='R')

s = tvm.create_schedule(R.op)
s[R].bind(s[R].op.axis[1], tvm.thread_axis("blockIdx.x"))  # map computation to gpu resources
tvm_fn = tvm.build(s, [A, B, R], target='cuda', target_host='llvm')  # generate and compile C++
tvm_fn.export_library('libmm.so')  # save to disk

tvm_fn = tvm.module.load('libmm.so')  # load from disk
my_mm_pytorch_fn = dlpack.to_pytorch_func(tvm_fn)  # package it as a PyTorch function

X = torch.randn(128, 256, device='cuda')  # allocate pytorch input tensors
Y = torch.randn(256, 32, device='cuda')  # allocate pytorch input tensors
Z = X.new_empty(128, 32, device='cuda')  # allocate pytorch output tensor
my_mm_pytorch_fn(X, Y, Z)  # call the tvm kernel

torch.allclose(X.matmul(Y), Z, atol=1e-04)  # compare tvm output with pytorch
```

Evaluation (1) - Character Level Language Modeling

dataset - Metric (bpc, lower is better)	Ours	SOTA (small model)
enwik8	0.999	1.02
text8	1.103	1.11

Large improvement

- Match SOTA result on the large model
- Requires sliding window with dilation on a few heads
- Largest train seqlen is 23k, evaluate on seqlen 32k
- fp16 and gradient checkpointing are important

Character Level Language Modeling - Details

- Many different ways to configure window sizes and dilation
  - increasing window size from 512 to 8192
  - dilation on layers 6-12 on two heads only
  - seqlen 32k
- Training in 5 phases, with every phase, 2x window size and seqlen and 0.5x LR
  - starting window sizes: 32 to 512
  - starting seqlen 2048
- Doubling window size has the most impact on bpc
  - At first, loss increases a lot then drops quickly (relevant for global attention later)
- Keep selfattention in fp32 because the model learns to use large attention scores

Pretraining and finetuning

Goal: a BERT-like for long-doc NLP tasks

Procedure to convert RoBERTa into Longformer

- Increase size of position embedding matrix. Initialize it by copying the first 512
- Continue MLM pretrianing on a corpus of long docs
- Copy q, k, v linear projections to get separate projections for global attention

This procedure can be easily applied to other models to convert them into Long

Pretraining - MLM results

Model	base	large
RoBERTa (seqlen: 512)	1.846	1.496
Longformer (seqlen: 4,096)	10.299	8.738
+ copy position embeddings	1.957	1.597
+ 2K gradient updates	1.753	1.414
+ 65K gradient updates	1.705	1.358

Finetuning on downstream tasks

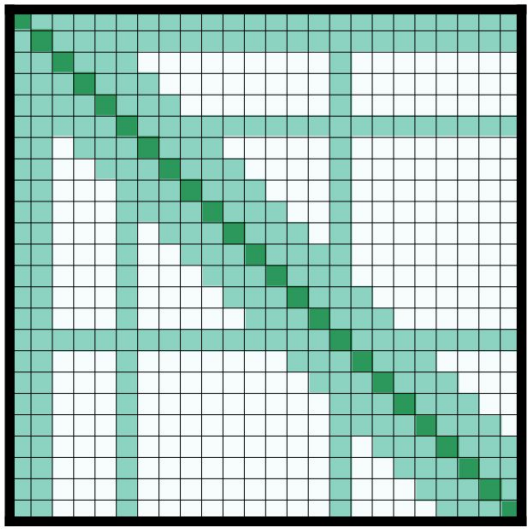
- HotpotQA -- Multihop reasoning and evidence extraction
- TriviaQA -- QA dataset from long Wikipedia articles
- WikiHop -- QA requiring combining facts spread across multiple paragraphs
- Coref -- Coreference chains across the document
- IMDB -- Document classification
- Hyperpartisan news -- Document classification

Finetuning on downstream tasks - Global attention

Classification: on CLS token

QA: on all question tokens

Coref: local attention only



Finetuning on downstream tasks - Results

	QA			Coref.	Classification	
Model	WikiHop	TriviaQA	HotpotQA	OntoNotes	IMDB	Hyperpartisan
RoBERTa-base	72.4	74.3	63.5	78.4	95.3	87.4
Longformer-base	**75.0**	**75.2**	**64.4**	**78.6**	**95.7**	**94.8**

Finetuning on downstream tasks - large model

SOTA on WikiHop and TriviaQA

Model	WikiHop	TriviaQA	HotpotQA
Current SOTA	78.3	73.3	**74.2**
Longformer-large	**81.9**	**77.3**	73.2

Competitive HotpotQA result

- Ours is simpler
- Better models use GNN of entities
  - Can we simulate it with global attention?

Model	ans.	supp.	joint
TAP 2 (ensemble) (Glaß et al., 2019)	79.8	86.7	70.7
SAE (Tu et al., 2019)	79.6	86.7	71.4
Quark (dev) (Groeneveld et al., 2020)	81.2	87.0	72.3
C2F Reader (Shao et al., 2020)	81.2	87.6	72.8
Longformer-large	81.3	88.3	73.2
ETC-large†	81.2	89.1	73.6
GSAN-large†	81.6	88.7	73.9
HGN-large (Fang et al., 2019)	82.2	88.5	74.2

Finetuning on downstream tasks - ablation

Model	Accuracy / Δ
Longformer (seqlen: 4,096)	73.8
RoBERTa-base (seqlen: 512)	72.4 / -1.4
Longformer (seqlen: 4,096, 15 epochs)	75.0 / +1.2
Longformer (seqlen: 512, attention: n^2)	71.7 / -2.1
Longformer (seqlen: 512, attention: window)	68.8 / -5.0
Longformer (seqlen: 2,048)	73.1 / -0.7
Longformer (no MLM pretraining)	73.2 / -0.6
Longformer (no linear proj.)	72.2 / -1.6
Longformer (no linear proj. no global atten.)	65.5 / -8.3

Table 11: WikiHop development set ablations

Conclusion

We should consider tasks beyond LM to develop and evaluate long models

Usage

```
model = transformers.AutoModel('allenai/longformer-base-4096')
```

Global attention is important and task specific

Follow our procedure to convert existing pretrained models into Long

Future work

`LongformerEncoderDecoder`

Better pretraining, other objective functions

Longer sequence

Better encoding of document structure

Global attention instead of GNN

More tasks; summarization, generation, IE

Multi-document

```
1 from transformers import AutoModel
2 model = AutoModel('allenai/longformer-base-4096')
```