Accurate and Efficient Attention with a Key-Value Cache

Anonymous ACL submission

Abstract

Attention networks have been proven effective in modeling long-term dependencies in sequence modeling, but the dependency range it can cover is limited by its high complexity. We propose a cache-based attention architecture to better and easier model long sequences. While reducing the complexity in long sequence modeling, the mechanism of selecting from cache also ensure the accuracy in picking historical information. When enlarging the dependency range of attention, our model is almost 1000 times more efficient than Transformer-XL. Experiments on language modeling shows that our model outperform traditional attention by a considerable margin. Our code and hyperparameters are available on $\langle link \rangle$.

1 Introduction

Long-term dependency is an important issue in Natural Language Processing (NLP). Many NLP tasks, such as Language Modeling, Reading Comprehension, Text Summarization, require the modeling of long-range dependency. Attention Models, especially BERT (Devlin et al., 2018), have achieved great success on a wild range of NLP tasks. Attention mechanism overcomes the weakness of Recurrent Neural Networks (RNNs) by directly connecting tokens beyond distance, which in the same time allowing the computation in parallel. However, the typical sequence length in BERT is 512, which is not enough for some tasks that need extra long-term dependency. Moreover, Attention mechanism suffers a complexity growing quadratically with sequence length , which restricts it to adopt a longer sequence length.

Many previous works have focused on how to modify attention to adapt it to the situation of modeling longer sequences. Some works add an additional fixed length memory unit to the attention mechanism. Transformer-XL (Dai et al., 2019) introduces the notion of recurrence among sequences into Transformer, and maintain a fixed length memory for each sequence, achieving a remarkable performance. Compressive Transformer (Rae et al., 2019) introduces an extra block to save several compressed sequence representation, this block along with the original memory block together work as the memory in the Transformer-XL. These models put all historical words into the computation of attention in a recurrent way, which faces almost the same high cost as original Transformer when enlarging the sequence length.

Other works sparsify the attention mechanism to reduce the high complexity. The Sparse Transformer (Child et al., 2019) used fixed mask to attend to roughly $\sqrt{l}$ (l is sequence length) positions in the memory. Reformer (Kitaev et al., 2020) used a local sensitive hashing to decide the postion to attend. Longformer (Beltagy et al., 2020) mixed fixed window attention with global attention to ease the computation of extra long sequence. The sequence length can be enlarged efficiently under this kind of sparse attention, but some possible relation between words may be cut due to the sparsity. The artificially designed sparse connection may possibly mismatch real long-term dependency between words. On the other hand, fully attention on longer sequence length does not definitely lead to better performance. As the sequence length become longer, more and more words are irrelevant to current meaning. It becomes harder for the fully-connected attention model to extract information from relevant words. This phenomenon both cause the performance to drop and the cost to increase. To avoid this phenomenon, we believe that historical words should be selectively dropped when utilizing them.

We propose a cache-based attention model to enable the operation of selection and at the same

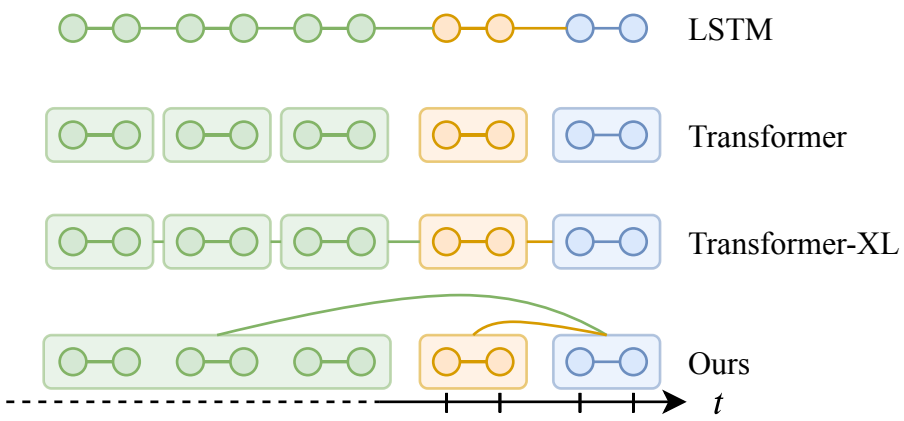

Figure 1: A comparison among models on modeling a sequence of given length.

time ensure the sequence length can be enlarged easily. The cache stores a large span of historical word representation in key-value pair. When selecting from the cache, we compute a query vector for each word and choose a fixed length of the most similar word representation from the cache. As a result, the model is capable of relieving the complexity of Transformer and at the mean time utilizing appropriate historical words. Since the selected length is fixed, our model can easily extend memory length to a large extent with a negligible increase in complexity. Compared to Transformer-XL, when enlarging memory length, our model can be 1000 times more efficient.

We evaluate our model on autoregressive word-level language modeling tasks. We demonstrate strong results on Penn Treebank, WikiText-2 and WikiText-103. We also show that our model can get further improvement through finetuning a Transformer-XL model, which ensures the compatibility of our model with existing Transformer models.

2 Related work

Language modeling is one of the most common tasks to verify the model's ability of modeling long-term dependency. In the past few years, many neural architectures were proposed to better encode the context (Bengio et al., 2003; Mikolov et al., 2010; Merity et al., 2016; Krause et al., 2016; Zilly et al., 2016; Dauphin et al., 2016; Al-Rfou et al., 2018; Dai et al., 2019). Methods to improve regularization, optimization and evaluation in language model are proposed as well (Zaremba et al., 2014; Koutník et al., 2014; Mikolov et al., 2014; Le et al., 2015; Grave et al., 2016; Yang et al., 2017; Krause et al., 2017; Brahma, 2018; Trinh et al., 2018).

To better make use of context, memory is often used as an additional input feeding to the neural network. Before the age of Transformer, memory had already been proven effective (Mikolov and Zweig, 2012). When it comes to Transformer, the component of memory helps Transformer to utilize context in a higher level (Dai et al., 2019; Rae et al., 2019). In order to modeling long sequences, many works relieve the high complexity of attention by sparsing it (Sukhbaatar et al., 2019; Tay et al., 2020b; Ye et al., 2019; Child et al., 2019; Zaheer et al., 2020; **?**). Some works choose to modify the computation of attention to speed up. The modification includes changing the order in matrix multiplication and using other functions or kernel methods to substitute the softmax function (Katharopoulos et al., 2020; Tay et al., 2020a; Choromanski et al., 2020).

Cache is used by some previous works to remember a larger capacity of information. Grave et al. (2016) introduced a continuous cache into language model to alter the output distribution based on the representation of historical identical words. Kuang et al. (2017) introduced a cache for words to improve Document-level Machine Translation. Tu et al. (2018) used a key-value cache to remember the translation history, in order to improve the consistency in Machine Translation. Cache can be of large capacity when containing words or representations and of high performance when extracting information. We believe that cache has more potential on extending the attention range while relieve the complexity.

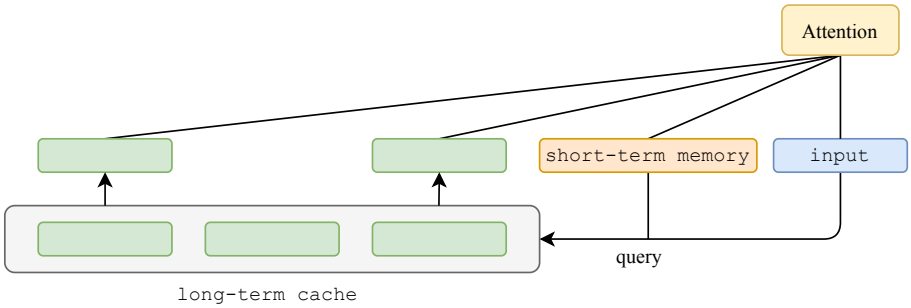

Figure 2: An overview of the whole process of our model. For an input sequence, the model first compute a query vector according to the input sequence and the short-term memory. After selecting from cache, the attention mechanism is applied among selected blocks, short-term memory and current inputs. Finally the cache is updated in preparation for next sequence.

3 Background

3.1 Language Model

A language model is to predict the probability of a sequence of words existing in the natural language. Given a sequence of words $\mathbf{x} = (x_1, \cdots, x_N)$, the goal of a language model is to estimate the joint probability $P(\mathbf{x})$, which can be computed auto-regressively by

$$P(\mathbf{x}) = \Pi_{t=1}^{N} p(x_t|\mathbf{x}_{<t}) \quad (1)$$

Indeed, what language models do is actually estimating x_{t+1} at time step t given previous words $x_1, \cdots, x_t$. In neural methods, we often encode the whole history into a fixed size hidden state $\mathbf{h}_t$, then the probability distribution of next word the network predicted is given by $\mathbf{p}_{t+1} = \text{softmax}(W\mathbf{h}_t)$, where W is a parameter matrix.

3.2 Multi-Head Attention

Multi-head Attention is proposed by Vaswani et al. (2017) to jointly attend to information from different representation subspaces. In practice, we derive a matrix Q composed of a set of queries, and packed together the keys and values as K and V as well. The computation of Multi-Head Attention is then

$$\text{MH-Attn}(Q, K, V) = Concat(h_1, \cdots, h_n)W^O$$
$$\text{where } h_i = softmax(\frac{(QW_i^Q)(KW_i^K)^\top}{\sqrt{d_k}})VW_i^V \quad (2)$$

Where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $W^O \in \mathbb{R}^{d_{model} \times nd_v}$ are projection parameters. n is the head number. d_k and d_v represent the dimension of key vector and value vector respectively. Traditional Transformer applies Multi-Head Self Attention, in which Queries, Keys and Values are exactly the same sequence. An easy way to add memory into Transformer is to substitute keys and values with longer sequence while maintain the query vector. Though easy, the extended sequence length in key and value is limited by the quadratic complexity.

4 Model

We proposed a cache-based Transformer model to better and easier capture long-term dependencies. As shown in Figure 1, the dependency range that the model can attend to through a direct path is much larger in our model than others. Our key idea is to save the representation of long-term memory in a key-value cache, and select part of them as the relevant history segment to take part in the attention computation. Thus we are able to extend the length of memory without a huge increase in complexity , and the cache-and-select design allows the attention mechanism to choose appropriate segment in the history, avoiding the inefficient full connection. Our model is composed of three parts. We will introduce how to select from cache in section 4.2, how to utilize selected segments to take part in the attention in section 4.3 and how to update the cache in section 4.4.

4.1 Short and Long-term Memory

In our model, the whole historical words are divided into two parts, we call the nearer one the short-term memory, and the farther one the long-term memory. We introduce a cache to store modeled representation of long-term memory in preparation for selectively using these words in later calculations. The short-term memory is considered so relevant as to definitely be chosen to take part in the calculation. In order to facilitate subsequent

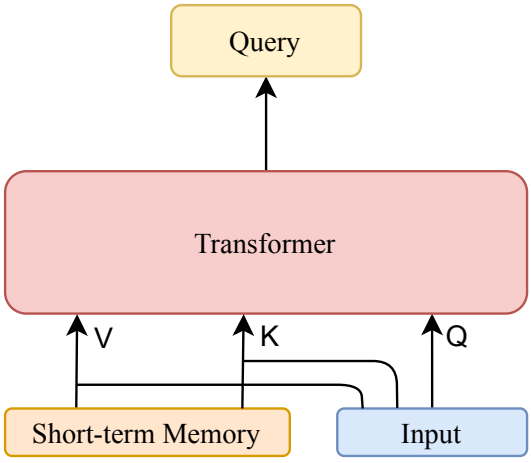

Figure 3: We reuse the Transformer model to compute query vectors for current words.

queries, the cache is designed in a key-value pair way, noted as

$$\mathcal{C} = \{\langle \mathbf{k}_i, \mathbf{V}_i \rangle\}_{i=1}^{N}, \tag{3}$$

where the key-value pair $\mathbf{k}_i \in \mathbb{R}^{d_k}$, $\mathbf{V}_i \in \mathbb{R}^{l\times(n+1)\times d_{model}}$ represents a modeled sequence of length l. As our model is based on an n-layer Transformer, the value $\mathbf{V}_i$ of the sequence is actually all layers including embedding layer of the Transformer, and the key $\mathbf{k}_i$ is a summary vector representing the meaning of the corresponding sequence. For simplicity, we use the top-layer representation of the last word of the sequence as the summary vector of the sequence. We have tried other ways of summarization but do not get better performance, we will state them in the experiment. We build a query based on current sequence and the short-term memory, then use the query to select k appropriate modeled sequences from the cache. We introduce them in detail as follows.

4.2 Selection from Cache

Assuming that the corpus is split into several sequences of length l. At time step t, current sequence is $\mathbf{X}_t = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l)$. Also assuming the short-term memory length is m, represents m tokens adjacent to current segment, noted as $\mathbf{S}_t = (\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m)$. For every word $\mathbf{x}_i$ in the current sequence, it should need different history fragments from the cache. So we shall query from cache for l times, get l different responses for each word. Recall that we choose the top-layer representation of the last word of the segment to be the key of a segment in the cache, and we expect a simple dot-product can well handle the problem of querying from cache. So we must assure that the key vector of segments in the cache and our derived query vector are in the same vector space. A direct solution is to reuse the Transformer encoder to precompute a representation of current meaning based on current sequence $\mathbf{X}$ and short-term memory $\mathbf{S}$, and take the representation at top layer as the query matrix $\mathbf{Q}$.

$$\begin{aligned} \mathbf{H}^{(1:n)} &= \text{Transformer}(\mathbf{S}, \mathbf{X}) \\ \mathbf{Q} &= \mathbf{H}^{(n)} \end{aligned} \tag{4}$$

where $\mathbf{H}^{(i)}$ represents the i^{th} hidden layer, they are computed just like that in Transformer-XL. Noted that the query matrix $\mathbf{Q} \in \mathbb{R}^{l\times d_{model}}$, it can be expressed as $\mathbf{Q} = (\mathbf{q}_1, \cdots, \mathbf{q}_l)$, where each $\mathbf{q}_i$ is a d_{model} dimension vector. And since $\mathbf{q}_i$ is exactly the final layer of the word x_i with right-blind mask, $\mathbf{q}_i$ is a reasonable representation of word $\mathbf{x}_i$ and its past information. The query matrix $\mathbf{Q}$ is in fact a collection of l query vectors of each word in current sequence.

Now that $\mathbf{q}_i \in \mathbb{R}^{d_{model}}$, the key of the j^{th} segment in the cache $\mathbf{k}_j \in \mathbb{R}^{d_k}$. We adopt $d_k = d_{model}$ in our experiment, so the key and the query are in same dimension, a direct dot-product can be applied. A linear projection can also be applied on the $\mathbf{Q}$ to map the query into dimension d_k if they are not in same dimension. We don't take it as default, but we discuss it in our experiment. For a certain query $\mathbf{q}_i$, we derive similarities between it and every segment in the cache, and then apply a top-k mechanism to get k most relevant segments and apply a softmax function to get the weight of selected segments.

$$\begin{aligned} \{\alpha_{ij}, \mathbf{C}_{ij}\}_{j=1}^{k} &= \text{topk}(softmax(\mathbf{q}_i^\top \mathbf{K})) \\ \mathbf{K} &= (\mathbf{k}_1, \cdots, \mathbf{k}_N) \end{aligned} \tag{5}$$

where $\mathbf{C}_{ij}$ are the stored value of selected segments from cache; α_{ij} is weight of each selected segment computed by softmax.

4.3 Attention Computation

Since we have chosen appropriate segments from the cache, the next step is to put them into the computation of attention. We concatenate the selected segments and the short-term memory to form an extended memory. The new extended memory contains information not only from important adjacent segment but also from relevant segments apart from long distance. The difference is that the value of

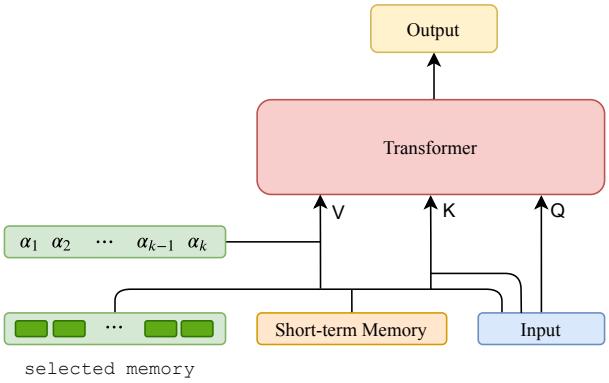

Figure 4: The process of Attention. Weights of each selected segments are multiplied to themselves when acting as values, while keys are just a concatenation of unweighted segments.

self-attention in our model is segment-weighted by the weight we just derived. To compute the r^{th} layer $\mathbf{H}^{(r)}$ in our model

$$\begin{aligned}
\mathbf{Q}^{(r-1)} &= \mathbf{H}^{(r-1)}\mathbf{W}_q^\top \\
\mathbf{K}_i^{(r-1)} &= [cat(\{\mathbf{C}_{ij}^{(r-1)}\}_{j=1}^{k}); \mathbf{S}^{(r-1)}; \mathbf{H}^{(r-1)}]\mathbf{W}_k^\top \\
\mathbf{V}_i^{(r-1)} &= [cat(\{\alpha_{ij}\mathbf{C}_{ij}^{(r-1)}\}_{j=1}^{k}); \mathbf{S}^{(r-1)}; \mathbf{H}^{(r-1)}]\mathbf{W}_v^\top \\
\mathbf{H}^{(r)} &= \text{MH-Attn}(\mathbf{Q}^{(r-1)}, \mathbf{K}^{(r-1)}, \mathbf{V}^{(r-1)})
\end{aligned} \tag{6}$$

where we use cat and $[\cdot;\cdot]$ indicates concatenation. The gradient of all representation gathered from cache and short-term memory is stopped. Following Transformer-XL, we use learnable relative embedding in our Multi-Head Attention layers. The modeling of the whole corpus is in a recursive way, we have just introduced the computation of one time step above. Next we will present how the recurrence is circulated.

4.4 Cache Update

Our cache deposits representations of N segments of length l in it. As long as we keep the selection number k, we can easily expand the memory range of the model by enlarge N with negligible extra cost in computation. Like any other cache, two important things are what to save and what to discard.

We update the cache in segment level. At each time step, we save the new modeled representation and discard an old segment in the cache by a certain strategy to maintain the cache in a fixed size. As we have mentioned, we take the top-layer modeled representation of the last word in the segment as the key and the whole representation as the value. Ideally, the segment we discard should be the least useful one in future modeling, but it is hard to measure the importance of segments from this point of view. So we just assume the more distance, the less importance, and take a simple First-In-First-Out way to update the cache. In practice, the segments are arranged in order of time in the cache, when a new segment come in, we simply discard the first segment, which is the oldest one. The segment that will be saved into cache is the first l words representation in the concatenation of short-term memory and current modeled sequence.

$$\begin{aligned}
\mathbf{V}_{new} &= [\mathbf{S}; \mathbf{H}][:l] \\
\mathbf{k}_{new} &= \mathbf{V}_l^{(n)} \\
\mathbf{S}_{new} &= [\mathbf{S}; \mathbf{H}][l:]
\end{aligned} \tag{7}$$

Again, $[\circ;\circ]$ indicates concatenation. $\mathbf{V}_{new}$ is the new value ready to be deposited and $\mathbf{k}_{new}$ is the new key of the value. $\mathbf{S}_{new}$ is the new short-term memory in next time step.

The Transformer-XL with a memory length of m has a maximum dependency range of m. The Compressive Transformer with a memory length of n_m and compressed memory length n_{cm} with a compression ratio c has a maximum dependency range of $n_m + c \times n_{cm}$. In their settings, c can not be too big so as to ensure the information retain during compression. Our model gives a dependency range of $m + N \times l$, and larger N brings larger dependency range with little extra cost.

4.5 Complexity

A key idea of our model is the selection from cache. It reflects both consideration of omitting useless information and controlling complexity. For a corpus divided into sequences each has a length of l, the computation of our model is composed of two parts, query process followed by Transformer attention computation, we call it attention process for short.

4.5.1 Query Process

In the query process, our model reuse the parameter of Transformer encoder to derive a query vector, and then use the query vector to search appropriate segments from the cache using a dot-product way. The computational complexity for a segment thus is

$$O(nl(l+m)d) + O(lNd) + O(lN\log N) \tag{8}$$

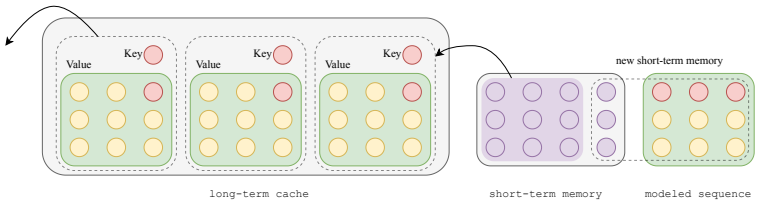

Figure 5: Update of the cache. Segments in the cache are in a fixed size l. We choose the representation of first l tokens in short-term memory and modeled current sequence to update into the cache. The oldest segment in the cache is discarded and the representation of remained tokens in short-term memory and current sequence act as new short-term memory.

Where m is length of short-term memory, n is layer numbers in the Transformer, d is the dimension of hidden state in the model, N is the number of segments in the cache.

The first term represents the computation of the query vector through Transformer layer, the second term indicates the dot-product weight computation. There are l queries, each compute a weight corresponding to N segments, and in each weight it is a d dimension dot product. The third term in the complexity is the sort of all N weights to gain top K important segments. As is always the case, $d >> \log N$,the sorting term of complexity can be ignored, the complexity of the query process can be simplified to

$$O(nld(l + m + \frac{N}{n})) \tag{9}$$

4.5.2 Attention Process

The complexity of attention process is exactly the same as Transformer-XL, except that the memory length is $m + kl$. So the complexity of attention process is

$$O(nld(l + m + kl)) \tag{10}$$

The total complexity of our model is the sum of two previous results:

$$O(nld(2l + 2m + kl) + ldN) \tag{11}$$

For fair comparison, we care about the computational complexity of different models to reach a same given attention range. We list them in Table **??**. In original Transformer-XL, if we enlarge the memory length with l, it will suffer an additional cost of $O(nl^2d)$. In our model, to get an additional l dependency range we can simply add N by 1. In that case, the complexity of attention process remains, and the extra cost is all from the query process at level $O(ld)$. In most cases, we choose train a Transformer Language Model with at least 12 layers and set an at least 80 words sequence length. So we can save almost **1000x** costs when enlarging the direct-link dependency range.

5 Experiments

We apply our model to various word-level datasets including Penn Treebank, WiKiText-103 and WikiText-2 and. We optimize all models with Adam. Follows Transformer-XL, we also adopted adaptive softmax and input representation. We clipped gradients to a max norm of 0.25. Other hyperparameters diverse for different benchmarks. We will introduce them separately in later section.

5.1 Penn Treebank

Penn Treebank is a traditional small-scale language modeling benchmark. It contains several short news passages with an average length of 355. we report the results on the word level Penn Treebank. We apply regularization tricks like variational dropout and weight average as the AWD-LSTM does. We train a 12-layer model with dimension 400 on single NVIDIA GTX 1080 Ti. We set our cache at size $N = 5$ and set the selection number at $k = 2$. We use a learning rate schedule with a 20,000-step warmup from 0 to 7e-4 and a cosine annealing process back down to 1e-6. We use 20000 warmup steps and the whole training process contains 250 epochs. As shown in Table 2, the result of our model surpass strong baselines by a remarkable margin, which verifies the superiority of our architecture.

5.2 WikiText-103

WikiText-103 is a common-used word-level language modeling benchmark with long-term dependency. It contains articles from Wikipedia with an average length of 3.6K tokens, so it is a good metric on the ability of modeling long-term depen-

Model	Complexity	Extending cost
Transformer	$O(n(l+m+Nl)^2d)$	$O(nl(2l+2m+N+1)d)$
Transformer-XL	$O(nl(l+m+Nl)d)$	$O(nl^2d)$
Compressive Transformer	$O(nl(l+m+\frac{Nl}{c})d)$	$O(\frac{nl^2d}{c})$
Ours	$O(nl(2l+2m+kl)d+Nld)$	$O(ld)$

Table 1: Computational complexity comparison among our model and other Transformer-based models. N is the capacity of our cache; l is the length of each sequence; m is the length of short-term memory; d is the dimension of hidden size and n is the number of Transformer layers. The first column lists the computational complexity of each model to reach an attention range of $l+m+Nl$; The second column lists the extra cost of each model to extend the attention range by l. The computational complexity of our model is irrelevant with N in quadratic attention term, while other models suffer unbearable quadratic extra cost when extend the attention range.

Model	**#Params**	**Valid. PPL**	**Test PPL**
RNN-based methods			
RNN-LDA + KN-5 + cache (Mikolov and Zweig, 2012)	9M	-	92.0
LSTM (Zaremba et al., 2014)	20M	86.2	82.7
variational LSTM (MC) (Gal and Ghahramani, 2016)	20M	-	78.6
CharCNN (Kim et al., 2016)	19M	-	78.9
Pointer Sentinel-LSTM (Merity et al., 2016)	21M	72.4	70.9
Tied Variational LSTM + augmented loss (Inan et al., 2016)	24M	75.7	73.2
LSTM + continuous cache pointer (Grave et al., 2016)	-	-	72.1
Variational RHN (tied) (Zilly et al., 2016)	23M	67.9	65.4
NAS Cell (tied) (Zoph and Le, 2016)	25M	-	64.0
skip-LSTM + dropout tuning (Melis et al., 2018)	24M	57.1	55.3
AWD-LSTM (Merity et al., 2017)	24M	60.0	57.3
AWD-LSTM-MoS (Yang et al., 2017)	22M	56.54	54.44
Mogrifier LSTM (Melis et al., 2019)	24M	51.4	50.1
AWD-LSTM + dynamic eval. (Krause et al., 2017)	24M	51.6	51.1
AWD-LSTM-MoS + dynamic eval. (Yang et al., 2017)	22M	48.33	47.69
Attention-based methods			
Transformer-XL (Dai et al., 2019)	24M	56.72	54.52
Transformer-XL (our imple.)			
Ours	23M	**56.36**	**54.16**

Table 2: Comparison with state-of-the-art results on Penn Treebank.

Model	#Params	Valid. PPL	Test PPL
RNN-based methods			
LSTM (Grave et al., 2016)	-	-	48.7
Temporal CNN (Bai et al., 2018)	-	-	45.2
GCNN-14 (Dauphin et al., 2016)	-	-	37.2
LSTM + cache (Grave et al., 2016)	-	-	40.8
4-layers QRNN (Merity et al., 2018)	151M	32.0	33.0
LSTM + Hebbian + cache + MbPA (Rae et al., 2018)	-	29.0	29.2
Attention-based methods			
Transformer + Adaptive inputs (Baevski and Auli, 2018)	247M	-	18.7
Transformer-XL base (Dai et al., 2019)	151M	23.1	24.0
Transformer-XL large (Dai et al., 2019)	257M	17.7	18.3
Transformer-XL + Dynamic eval.(Krause et al., 2019)	257M	15.8	16.4
Compressive Transformer (Rae et al., 2019)	-	16.0	17.1
Routing Transformer (Roy et al., 2020)	-	-	15.8
Transformer-XL (Our imple.)	166M		
Ours base	166M	23.93	24.97
Ours base (finetune)	166M	21.9	22.8

Table 3: Comparison with state-of-the-art results on WikiText-103

dencies. We finetune a pre-trained Transformer-XL model with 16 layers and set the sequence length to 150 , short-term memory length to 150. We set our cache at size $N = 5$ and set the selection number at $k = 2$. We use a learning rate schedule with a 20,000-step warmup from 0 to 3e-4 and a cosine annealing process back down to 1e-6. We finetune our model with 6 NVIDIA GTX 1080 Ti. Table 3 shows that we obtain a perplexity of XX based on a pre-trained model with a perplexity of XX. The experiments shows that our model has the potential to gain further improvement based on well-trained Transformer language model.

5.3 WikiText-2

We also conduct experiments on WikiText-2. The size of WikiText-2 is roughly twice as the size of PTB. It contains articles from Wikipedia with nearly 33k tokens in average. Articles in it is in a non-shuffled order, with dependencies across articles remained. The settings of the model is just the same as that in the experiment of PTB. The results are shown in Table 4. Transformer-XL did not report results on WikiText-2. The results of Transformer-XL in the table is based on our implement of Transformer-XL. We get State-of-the-Art result on WikiText-2 among all attention language models without other evaluation techniques such as dynamic evaluation.

5.4 Variation of Model

We have tried different summary methods to derive the key vector of a segment in the cache. Reasonably, the key vector should be computed based on the top layer of sequence representation. By default we simply use the representation of last word as the key. We also tried max / mean pooling and parameterized weighted sum. For the computation of query vector, as we have mentioned before, we have tried adding an additional linear projection with an activation function. Also we have tried simply use the word embedding as the query vector, proving the necessity of reusing the Transformer model. We have also tried other techniques in updating cache. We tried discarding the least use segment instead of the FIFO strategy, but the experiment shows that the more complicated design does not lead to a better performance.

5.5 Analysis

We compute perplexity of buckets of words classified by word frequency. The observation is carried out on the validation set of WikiText-2. We set 4 word frequency bucket, that is 0-100, 100-1K, 1K-10K, >10K. We hold the following hypothesis: Words that occur frequently such as stop words

Model	#Params	Valid. PPL	Test PPL
RNN-based methods			
Variational LSTM + augmented loss (Inan et al., 2016)	28M	91.5	87.0
LSTM + continuous cache pointer (Grave et al., 2016)	-	-	68.9
skip-LSTM (Melis et al., 2017)	24M	69.1	65.9
AWD-LSTM (Merity et al., 2017)	33M	68.6	65.8
AWD-LSTM-MoS (Yang et al., 2017)	35M	63.88	61.45
Mogrifier LSTM (Melis et al., 2019)	24M	57.3	55.1
Attention-based methods			
Transformer-XL (our imple.)	33M	60.19	57.87
Ours	33M	**59.26**	**56.91**

Table 4: Comparison with state-of-the-art results on WikiText-2.

query method	summary method	update method	**Test PPL**
pre-compute	last state	FIFO	56.91
+ linear word embedding			
	max pooling		
	mean pooling		
	weighted sum		
		discard most useless	

Table 5: Model Variation. We have tried different method on the process of query, summarization and update. The experiment is carried out on WikiText-2.

	$<$100	100 - 1K	1K - 10K	$>$10K	All
LSTM					
Transformer-XL					
Ours					

Table 6: WikiText-103 test perplexity computed within different word frequency bucket.

often need little long-term information; On contrary, words that seldom occur are always entities or words with special meaning, which maintain more relation with context. As shown in Table 6, our model's perplexity of low-frequency bucket is much lower, which shows our model's great improvement in modeling rare words.

5.6 Long term efficiency

Khandelwal et al. (2018) proposed Effective Context Length (ECL) as an automatic metric to evaluate the model's ability of utilizing context. It is defined as the longest context length that the model can utilize to get a lower perplexity over a fixed threshold. We compare the ECL of our model, Transformer-XL and LSTM. The ECL of our model is longer than that of the other models, indicating our model's ability to capture longer-term dependency.

6 Conclusion

In this paper we propose a notion of cache so as to efficiently extend the dependency length of traditional Transformer architecture. Our cache-select scheme enable the model to look into a longer range of history and the size of cache can be easily enlarged without worrying a huge increase in computational cost. The selection scheme also allows the model to omit irrelevant words, relieve the overfull connection of attention when modeling long sequences.

We believe that attention mechanism has more potential power in modeling dependencies. When it comes to extra long dependencies, attention faces a problem of extra high complexity and unnecessary full connections. Our cache-based model is a kind of way to sparse the connections as well as to reduce the complexity. However, our segment-wise selection assumes that words in the same segment hold relatively alike meanings, which is not always the case. Selecting the segment as a whole to take part in the following attention will inevitably introduce information of no effect. Ideally, segments in the cache should hold a complete meaning separately, and not contain irrelevant words so that the query can be highly effective. But so far we do not have a technique to detect the "meaning boundary". No matter we haven't achieved, cache and selection are two of the key methods in reforming the attention mechanism when applying them to long-range problems. Cache-based attention methods are still a research field with high potential.

Acknowledgments

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-Level Language Modeling with Deeper Self-Attention.

Alexei Baevski and Michael Auli. 2018. Adaptive Input Representations for Neural Language Modeling. *7th International Conference on Learning Representations, ICLR 2019*.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. CONVOLUTIONAL SEQUENCE MODELING REVISITED. Technical report.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Siddhartha Brahma. 2018. Improved Language Modeling by Decoding the Past. pages 1468–1476.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. Technical report.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Rethinking Attention with Performers.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. Technical report.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, pages 1–9.

	Valid. PPL	ECL		
		r=0.1	r=0.5	r=1.0
LSTM				
Transformer-XL				180
Ours				220

Table 7: ECL comparison.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-Aware Neural Language Models. In *AAAI*, pages 2741–2749.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer.

Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. 2014. A Clockwork RNN.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2017. Dynamic Evaluation of Neural Sequence Models. *35th International Conference on Machine Learning, ICML 2018*, 6:4320–4329.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2019. Dynamic Evaluation of Transformer Language Models.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. 2016. Multiplicative LSTM for sequence modelling. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*.

Shaohui Kuang, Deyi Xiong, Weihua Luo, and Guodong Zhou. 2017. Modeling Coherence for Neural Machine Translation with Dynamic and Topic Caches.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units.

Gábor Melis, Charles Blundell, Tomáš Kočiský, Karl Moritz Hermann, Chris Dyer, and Phil Blunsom. 2018. Pushing the bounds of dropout.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*.

Gábor Melis, Tomáš Kočiský, and Phil Blunsom. 2019. Mogrifier LSTM. *arXiv*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. 2014. Learning Longer Memory in Recurrent Neural Networks.

Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. *SLT*, 12(234-239):8.

Tomas Tomáš Mikolov, Martin Karafiát, Lukáš Lukas Burget, Jan Černock\'y, Sanjeev Khudanpur, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, September, pages 1045–1048.

Jack W Rae, Chris Dyer, Peter Dayan, and Timothy P Lillicrap. 2018. Fast Parametric Learning with Activation Memorization. *35th International Conference on Machine Learning, ICML 2018*, 10:6739–6755.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. 2019. Compressive Transformers for Long-Range Sequence Modelling.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient Content-Based Sparse Attention with Routing Transformers. *arXiv*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive Attention Span in Transformers. pages 331–335.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking Self-Attention in Transformer Models.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020b. Sparse Sinkhorn Attention. *arXiv*.

Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. 2018. Learning Longer-term Dependencies in RNNs with Auxiliary Losses.

Zhaopeng Tu, Yang Liu, Shuming Shi, and Tong Zhang. 2018. Learning to Remember Translation History with a Continuous Cache. *Transactions of the Association for Computational Linguistics*, 6:407–420.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2017. Breaking the softmax bottleneck: A high-rank RNN language model. *arXiv preprint arXiv:1711.03953*.

Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. 2019. BP-Transformer: Modelling Long-Range Context via Binary Partitioning. *arXiv*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big Bird: Transformers for Longer Sequences.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2016. Recurrent Highway Networks.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

A Appendices

Appendices are material that can be read, and include lemmas, formulas, proofs, and tables that are not critical to the reading and understanding of the paper. Appendices should be **uploaded as supplementary material** when submitting the paper for review. Upon acceptance, the appendices come after the references, as shown here.

LaTeX-specific details: Use `\appendix` before any appendix section to switch the section numbering over to letters.

B Supplemental Material

Submissions may include non-readable supplementary material used in the work and described in the paper. Any accompanying software and/or data should include licenses and documentation of research review as appropriate. Supplementary material may report preprocessing decisions, model parameters, and other details necessary for the replication of the experiments reported in the paper. Seemingly small preprocessing decisions can sometimes make a large difference in performance, so it is crucial to record such decisions to precisely characterize state-of-the-art methods.

Nonetheless, supplementary material should be supplementary (rather than central) to the paper. **Submissions that misuse the supplementary material may be rejected without review.** Supplementary material may include explanations or details of proofs or derivations that do not fit into the paper, lists of features or feature templates, sample inputs and outputs for a system, pseudo-code or source code, and data. (Source code and data should be separate uploads, rather than part of the paper).

The paper should not rely on the supplementary material: while the paper may refer to and cite the supplementary material and the supplementary material will be available to the reviewers, they will not be asked to review the supplementary material.