Memory transformer with hierarchical attention for long document processing

1st Arij Al Adel
[1] *Moscow Institute of Physics and Technology,*
Dolgoprudny, Russia
[2] *Aleppo University*
Aleppo, Syria
arij.aladel@gmail.com

Mikhail S. Burtsev
[1] *Artificial Intelligence Research Institute,*
[2] *Moscow Institute of Physics and Technology*
Moscow, Russia
burtcev@airi.net

2021 International Conference Engineering and Telecommunication (En&T) | 978-1-7281-8390-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ENT50460.2021.9681776

***Abstract*—Transformers have attracted lots of interest from the researchers. Up to now, transformers achieved state-of-the-art results in a wide range of natural language processing tasks such as different sequence modeling tasks like language understanding, text summarization and translation, and definitely more transformers to come. Still, transformers has their limitations in the tasks requiring long document processing. This paper introduces a new version of transformer, a Sentence level transformer with global memory pooling and hierarchical attention to cope with long text. We replace self-attention of vanilla transformer with multi-head attention between memory and a sequence, and also add a decoder sequence selector on the top of the encoder output. In our architecture sentences are encoded in parallel and then summarized with soft-attention on every decoding step. Proposed model was validated in machine translation task. We hypothesize that attaching memory slots to each sequence improves the quality of translation, besides tuning the model on context-aware data set by using pre-trained sequence-level weights will help to get more precise translation and promote translating long documents. Results show that extending each sentence with a memory slot and employing the attention over the encoder outputs improves translation results.**

***Index Terms*—memory transformers, memory slot, attention, long document, translation, context aware, context agnostic**

I. Introduction

Transformer was primarily introduced as a sequence-to-sequence model for translation task [1]. Since then, transformer-based architectures achieved and holds the state of the art results in different sequence modeling tasks like language understanding, text summarization and many other artificial intelligence areas like computer vision, and audio processing. There are two basic components that contributed to the transformer success, the self attention and the position-wise feed forward network (FFN). Unnormalized self-attention is usually calculated as:

$$Attention(Q, K, V) = softmax(\frac{\mathbf{QK}^T}{\sqrt{d}})\mathbf{V}, \qquad (1)$$

where query $\mathbf{Q}$, key $\mathbf{K}$ and value $\mathbf{V}$ are the linear projection of the same input $\mathbf{X}$:

$$\mathbf{Q} = \mathbf{X}W^q, \mathbf{K} = \mathbf{X}W^k, \mathbf{V} = \mathbf{X}W^v.$$

However, this definition of self-attention has a substantial shortcomings. Because update of every vector takes into account only on pairwise similarities, self-attention can't model a prior knowledge of relationships between different parts of a long text. Suppose we have a long document as input to the transformer, and as the transformer input is limited in length, we have to divide this document into equal parts during processing. Feeding two consequent parts to the transformer make it unable to reason about the relationships between words of these two parts. Our proposed model tries to solve this problem by adding a memory slot which can be one or more of any reserved special tokens to every sequence. Every sequence has its independent memory slot.These memory slots can store summary of the sequence to be attended by an inter-sequence level attention during decoding.

Since improving methods on self attention such as lightweight attention (e.g. sparse attention variants) [2], [3] are not a complimentary ticket [3]; using sparse attention requires adding more layers, sparse attention was not used in the model and in our experiments full attention was used.

Self-attention is not the only component in the transformer and other parts such as FFN which constitutes two-thirds of it, also contribute to transformer success in different tasks. Many studies were performed to explore FFN modifications and it's alternatives in transformer models.

II. Related work

In the past three years, there were a few works that tried to add memory to the transformer structure in different ways. [4] presents a model that consists purely from augmented self attention layers they called it all-attention layer, these layers boosted by additional persistent memory in key and value vectors' pool. Authors explained that feed forward layers are equivalent to attention layer if replacing the RELU function by a soft-max function and removing the biases. They take this advantage and add vectors not conditioned on the input to apply attention on it beside the sequence of input vectors. These additional vectors play memory role; catch information that does not depend on the immediate context. This model was used to solve language modeling problem. Adding memory slots into keys-values of attention founds

its way into image captioning and the results was shown in [5], which has close idea to the previous work [4]; the encoder has Memory-Augmented Attention, in which keys and values are augmented by memory slots, these memory slots are responsible of encoding the previous knowledge and can be updated via stochastic gradient descent, the modification in this model is not limited to encoder but to the decoder. Authors come up with a meshed attention to exploit all encoding layers to produce a response. [6] produces a memformer model which is simply a traditional transformer vanilla equipped with a memory system, the memory system is composed of two components ; memory cross attention and memory slot attention. Memory cross attention is responsible for retrieving the information from the memory for the current step of encoding. Here the key - value is just the retrieved memory whilst the query is the current segment text to encode this way the input will attend just over key-value memory. Memory slot attention is for updating the memory to be used for encoding the next input of the encoder. In this structure the modification is in encoder side, which is meant of encoding the input and updating the memory. This model can handle denoising objectives and language modeling tasks. Adding external memory to this model led to an exorbitant memory cost that is why a new optimization scheme, memory replay backpropagation (MRBP) was also introduced. The concept of global attention or using global tokens was also studied by [3], [7]–[11]. Star-Transformer [7] use the graph theory to build the attention mechanism. Star-Transformer includes just one global node to connect every two non-adjacent nodes, besides it sparsifies the attention by using star-shaped structure. In this way Star-transformer maintain two kinds of communication between nodes; global connections and ring connections. It was called star cause the attention has star shape in which the global node is the domain center of gathering and distributing the information. Ring connections allow to collect information from neighbors. [8] and [9] were developed in parallel and have similar sight of adopting global nodes to preserve the global attentions between nodes. Longformer [8] in design is close to star-Transformer; whilst Star-transformer uses fixed-size window attention around each token and global attention, Longformer incorporates in addition dilated sliding window. On the other hand ETC [9] is similar to Longformer in using global and local attention but differs in using relative position embedding; Longformer uses it just for inference. Big-bird [3] was the successor of ETC; and it was explored over more tasks; ETC handles MLM with whole word masking and Contrastive Predictive Coding CPC to predict subsequent inputs in latent space Big-bird handles in addition question answering and document summarization. Star-Transformer was verified over three NLP tasks including Text Classification, Natural Language Inference, and Sequence Labelling. Memory transformer [10] called these global tokens memory tokens also [11] presents similar idea using sparse attention.

In addition to the attention study line in transformers, there is another interesting and promising research direction dedicated for using Feed forward network in transformers and its variants [4], [12], [13] some works studied using different activation functions [14], [15] instead of just using Rectified Linear Units (ReLU) activation for non-linearity in between the two FFN layers. Another works investigate replacing FFN with another layers for example [12] introduced product-key memory layers and replaced FFN with it in specific transformer layers. This study showed how using product-key memory improves the transformer performance with negligible computation overhead. [13] has investigated using product-key memory in pre-trained language models (PLMs) and on contrary to [12] reveals that most of memory slots remain outdated during training PKM-augmented models, that means applying PKM directly does not give the expected results and they propose that both of FFN and PKM are necessary for the pre-training. On the other hand [4] proposed a model constitutes just from attention layers, augmenting the attention with persistent memory vectors and removing the FFN layers. their model was examined on standard character and word level language modeling benchmarks. Our work is similar to previous works [3], [10] in using special tokens as global memory slots but different in usage way and attention used; we are using full attention connection.

III. Model

Given a corpus of tokens $\mathbf{x} = (x_1, x_2, ..., x_n)$ splitting it into K equal segments where the last segment is the source that requires a response and the previous $k-1$ segments represent the context. This kind of formulation is suitable for translation tasks, question answering tasks with context, and reading comprehension tasks. This paper will show the results of applying the model for translation task using the same data sets used by [16]. T5 transformer is the baseline to compare with in our study.

Fig. 1 represents the proposed model. Attaching memory slots to each sequence in the encoder side of the transformer and adding sequence selector between encoder and decoder. We use T5 transformer vanilla from hugging face, and we make three main changes: attention in encoder to handle added memory slots to each sequence, we add sequence selector between the encoder and the decoder, and modify cross attention in the decoder of the model. The model is comprised of three main components.

A. Encoder

Each encoder layer consists of two sub layers attention layer and Feed forward layer, these two layers differ from these layers in the T5 transformer. The encoding process has the following steps: input of encoder could be one sequence or source sequence with n previous sequences as context. In this case the model should learn which of these sequences is the source and which of these sequences is from the context sequences. We will see if the context sequences could help to improve the translation or not. If the input of encoder consists of more than just one sequence then it will be divided into sequences each sequence will has its own slot memory. Randomly adding fixed number of memory tokens we call

it slot or we simply add special tokens as memory tokens to avoid random initialization. We add slot memory to each sequence inside the encoder, and apply attention between the memory slots and the sequences . Every slot has access to all other slots, and only to token representations of related sequence. Also Every token in a sequence has access to all tokens of the same sequence (inner sequence attention) and all slots. with noting that sequences are encoded in parallel. At the end of encoding process we will have two hidden states for representing each of memory slots and their related sequences. For positional encoding we pay attention to encode each slot memory with its sequence. No positional encoding between sequences was applied, this is the memory slots mission.

1) Positional encoding: There was not any kind of positional encoding between consequent input sequences. T5 default positional encoding was used instead in a way that take into the consideration the position between each memory slot and its related sequence. We left the mission of positional encoding between sequences to the memory slots and sequence selector as mentioned before.

2) Multi-head encoder cross attention: We call this attention MemAttention, its input consists of $X^{mem+sent}$ which represents each memory slot with its related sentence, and will be separated into two separate queries inside our attention. The input also consists of $X_c^{mem+sent}$ of dimension d_k for each head, this is the sentence prefixed by all memory slots of all sentences from the same input sample in the same order of the sample sentences, and this represents the keys and values to use in MemAttention. Inside MemAttention memory slot is separated from its related sentence to get two separate queries, keeping the key-value without any change then calculate attention for each of memory slot and related sentence independently. The output of attention layer is:

$$att = LN(X^{mem+sent}) + MemAttention(X^{mem+sent}, X_c^{mem+sent}, X_c^{mem+sent}), \tag{2}$$

where MemAttention has two variants to calculate, as explained in section IV.

3) Feed forward network: after attention we need to apply FFN as T5 transformer vanilla but in this structure applying FFN have two variants symbolically I called them (1FFN and 2FFN), they are explained in section IV, to remember X here is the output of previous attention layer which is the concatenation between memeory slot representation and the related sentence; $X = A = [A^{mem}_{mems+sent}; A^{sent}_{mems+sent}] \in R^{(c+L)*d_{model}}$ the dimension of hidden layer of FFN is $d_{ff} = 2048$

B. Sequence selector

The output of encoder is separated to memory slots representation H^{mem} and related sentence representation H^{sent}. To recall, before encoding we have reshaped the memory representation sentence representation input embedding, that is why after encoding we return the original batched shapes; batched memory representation after encoding $\in R^{b*k*c*d_{model}}$. The

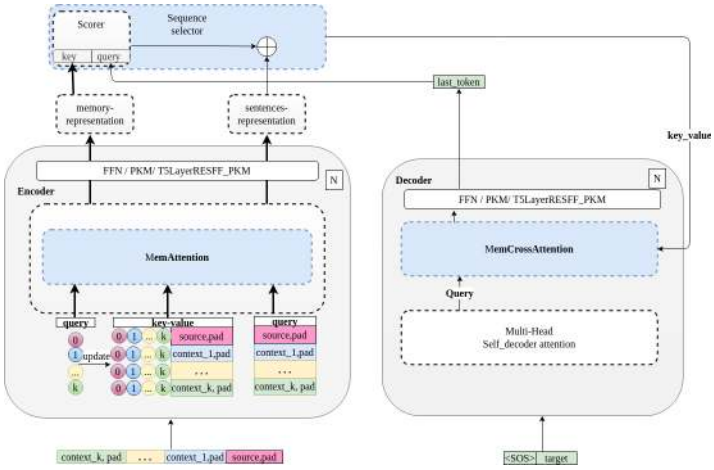

Fig. 1. Modified T5 transformer vanilla with added memory to each sequence in the encoder. Encoder input are tokens of padded sequences. Decoder input are target tokens.

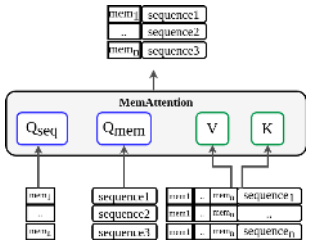

Fig. 2. Attention between memory slots, sequences and the concatenation between all memory slots ans sequences.

batched sentences representation is $\in R^{b*k*L*d_{model}}$; b is the batch size, k is the number of input sentences including the source and context, c is the number of memory tokens in the memory slot and L is the length of each sentence in the input. The inputs of the sentence selector are three; query comes from decoder(model output), key memory representation from encoder and sentences representations from encoder. To calculate the weighted score between the decoder output and memory slots resulted from the encoder and choose the next token to be generated from one of source-context sequences and it works after projecting the selector inputs as following; first calculate the score between decoder and memory slots then product each sequence with its score we get sequence representations to be lately summed up and get one vector representation of source size for each token in the target. This process can be expressed as:

$$S = softmax(Query * Key^T), \tag{3}$$

where $Query = query * W^q; W^q \in R^{d_{model}*(c\_d_{model})}, c\_d_{model} = c * d_{model}$, $Key = key * W^k; W^k \in R^{d_{model}*d_{model}}$.

This means that every target token has k scores for each sentence in the corresponding input. Then we product each sentence with its score then sum up all scored sentences

representations to get one vector $\in R^{L*d_{model}}$ for each target token.

$$Z = \sum_{1<=i<=k} Sent_i * S_i, \tag{4}$$

where $Sent_i = Sent_i * W^{sent}; W^{sent} \in R^{d_{model}*d_{model}}$.

C. Decoder

Since the key-value vectors are not more coming directly from the encoder but from the sequence selector which has one more additional dimension than the encoder output; this is because for each target token we have one vector not like T5 transformer which use just one memory representation to generate all target tokens. According to that cross attention in the decoder has been modified to handle this issue.

1) Decoder Multi-head self attention: The same as T5 self attention

2) Multi-head encoder-decoder cross attention: This is the bottleneck of our model and is the future study topic. Apply attention between sequence selector output and decoder self attention output. We can express it as following:

$$MHA(Q, K, V) = diag(concat(head_1, .., head_h)W^O), \tag{5}$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.
$W_i^Q, W_i^K, W_i^V \in R^{d_{model} \times d_{head}}$ are the parameter matrices of linear transformation, $Q \in R^{T \times d_{head}}$ for each $head_i$ and it is the previous layer output in the decoder. K and V are the sequence selector output $\in R^{T \times L \times d_{head}}$ for each head. $W^O \in R^{d_{model}*d_{model}}$.

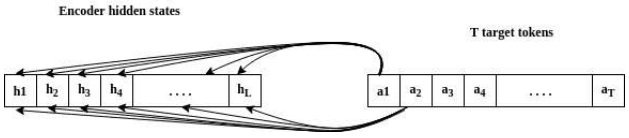

Fig. 3. T5 Transformer Encoder Decoder Attention: the queries come from the previous decoder layer, and the keys and values come from the output of the encoder, where every position in decoder attends to all positions from encoder

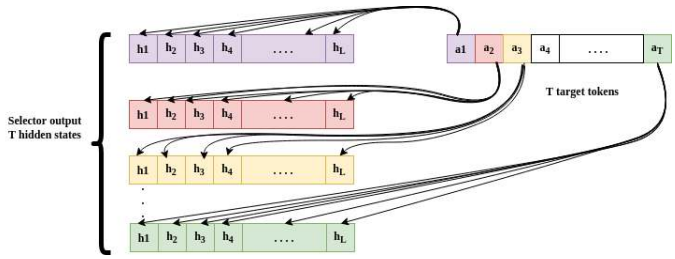

Fig. 4. T5Mem Encoder Decoder Attention: in our model where the queries come from the previous decoder layer,and the keys and values come from the output of the sequence selector, where every position in decoder depends on all positions of new coming hidden states from sequence selector.

As it clear the key difference between the cross attention in our model Fig. 4 and the T5 transformer cross attention Fig. 3 is that each decoded token in our model depends on the new hidden states that are generated depending on the previous generated token from the model, while in the T5 transformer the last hidden state from the encoder is used as a memory or context to generate all the response tokens.

IV. Further architecture design

To explore wider spectrum of design choices, and get more insight of how different design options can affect the model performance we tried many variants and reported the results.

To understand the variants we will start by explaining the MemAttention work concept. In MemAttention we have two kinds of implementations to explore the effect of last linear layer on attention performance. First option was using one output linear layer for the memory slot, sentence concatenated attentions. Starting with the MemAttention input which consists of $X^{mem+sent}$ which represents each memory slot with its related sentence, and will be separated into two separate queries inside our attention, and $X_c^{mem+sent} \in R^{(c+L)*d_k}$ for each head, this is the sentence prefixed by all memory slots of all sentences from the same input sample in the same order of the sample sentences, and this represents the keys and values.

Recall that MemAttention is calculated according to III-A2 where $X^{mem+sent}$ is the concatenation of memory slot and its related sentence, which will divided into two queries inside the attention one is the memory slot query X^{mem} and the other is the sentence query X^{sent} and $X_c^{mem+sent}$ is the keys values representation, projection matrices are $W^{Q_{mem}} \in R^{d_{model}*d_{model}}$ for memory query, $W^{Q_{sent}} \in R^{d_{model}*d_{model}}$ for sentence query $W^K \in R^{d_{model}*d_{model}}$, and $W^V \in R^{d_{model}*d_{model}}$. Linear projections:

$$Q_{mem} = X^{mem} * W^{Q_{mem}} \in R^{c*d_{model}},$$

$$Q_{sent} = X^{sent} * W^{Q_{sent}} \in R^{L*d_{model}},$$

$$K = X_c^{mem+sent} * W^k \in R^{(k*c+L)*d_{model}},$$

$$V = X_c^{mem+sent} * W^v \in R^{(k*c+L)*d_{model}}.$$

Scores calculation is common between the two variants, to calculate each of $score_{mem}$ and $score_{sent}$ we need to calculate the common position_bias and sum it with the concatenation of $score_{mem}$ and $score_{sent}$; this guarantees positional embedding of each memory slot to be with its related sentence.

$$score_{mem} = Q_{mem} * K^T \in R^{c*(k*c+L)}, \tag{6}$$

$$score_{sent} = Q_{sent} * K^T \in R^{L*(k*c+L)}, \tag{7}$$

concatenate scores to sum up with the computed positional bias so the final concatenated score will be:

$$score = [score_{mem}; score_{sent}] + position\_bias, \tag{8}$$

where $score \in R^{(c+L)*(k*c+L)}$. After that separate them again to continue calculations.

A. 1Linear

Here as shown in Fig. 5 just one output layer $W^O \in R^{d_{model}} * R^{d_{model}}$ after concatenating the memory slot attention and sentence attention; for short we will call $T5MemAttention(X^{mem+sent}, X_c^{mem+sent}, X_c^{mem+sent})$ as A:

$$A = att * W^O, \tag{9}$$

$$att = [att_{mem}; att_{sent}], \tag{10}$$

$$att_{mem} = [head_{i_{mem}}; .., headn_{mem}], \tag{11}$$

$$att_{sent} = [head_{i_{sent}}; .., headn_{sent}], \tag{12}$$

$$head_{i_{mem}} = softmax(score_{mem}) * V_{head_i}, \tag{13}$$

$$head_{i_{sent}} = softmax(score_{sent}) * V_{head_i}, \tag{14}$$

where $att \in R^{(c+L)*d_{head}}$, $head_{i_{mem}} \in R^{c*d_{head}}$ and $head_{i_{sent}} \in R^{L*d_{head}}$

B. 2Linear

Here as shown in Fig. 6 we have two output layers: $W^O_{mem} \in R^{d_{model}*d_{model}}$ and $W^O_{sent} \in R^{d_{model}*d_{model}}$: and the final MemAttention is calculates as follows:

$$A^{mem}_{mems+sent} = att_{mem} * W^O_{mem}, \tag{15}$$

$$A^{sent}_{mems+sent} = att_{sent} * W^O_{sent}, \tag{16}$$

$$A = [A^{mem}_{mems+sent}; A^{sent}_{mems+sent}], \tag{17}$$

c is the number of memory tokens in the memory slot, L the length of source sequence and each of context sequences if exist, k is the number of input sequences, d_{head} is the head dimension = 512/num_heads = 64, d_{model} =512, and number of heads is 8.

C. 1FFN

Use one Feed forward after MemAttention.

D. 2FFN

Here we separate the memory representation and the sentence representation and pass each to separate feed forward network layer:

$$A^{mem}_{mems+sent}; A^{sent}_{mems+sent} = A,$$

then again concatenate them as one output of the encoder. Memory sentence representation results from the encoder $H \in R^{(k*bz)*(c+L)*d_{model}}$; bz is the input batch size feed into the encoder.

V. Used versions

1) v1: uses 1Linear Fig. 5 inside the MemAttention to handle both memory slots and related sequences but two separated Feed Forward Fig. 8 after MemAttention; one for memory slots and the other for sequences representations.

2) v2: uses 1Linear Fig. 5 inside the MemAttention to handle both memory slots and related sequences and one Feed Forward Fig. 7 after MemAttention; for both memory slots and the other for sequences representations.

3) v3: uses 2Linear Fig. 6 inside the MemAttention to handle memory slots and related sequences independently, and two separated Feed Forward Fig. 8 after MemAttention; one for memory slots and the other for sequences representations, we should mention that this was the main proposed design of the model.

4) v4: uses 2Liner Fig. 6 in MemAttention and just one feed forward Fig. 7 after attention.

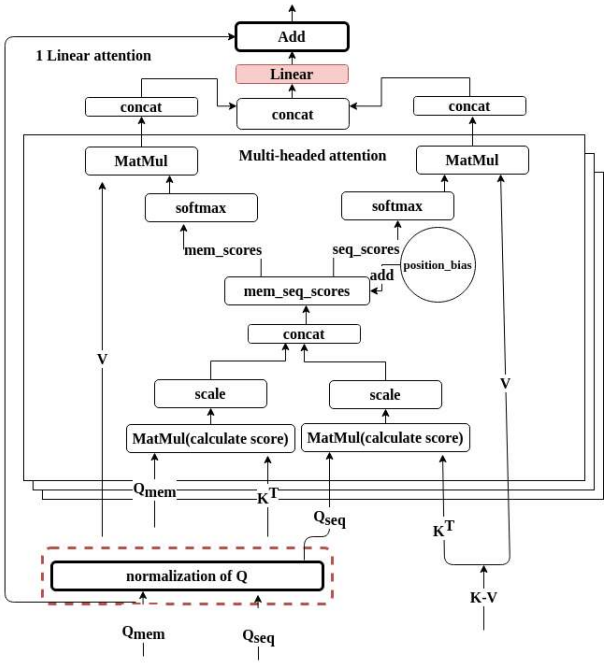

Fig. 5. 1Linear attention: MemAttention with one linear layer for both processed memory tokens and sequence tokens.

Results are reported in section VII

VI. Experiments

A. Experiment Settings

SentencePiece tokenizer was trained on training data files[1]for both languages Russian and English [17] and was used to encode text as WordPiece tokens with source and

[1]https://www.dropbox.com/s/5drjpx07541eqst/acl19_good_translation_wrong_in_context.zip?dl=0

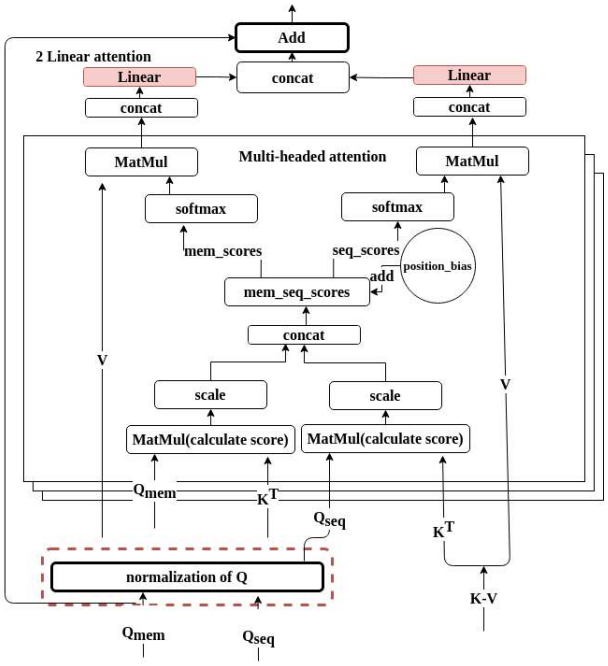

Fig. 6. 2Linear attention: MemAttention with two separate linear layer for both processed memory tokens and sequence tokens.

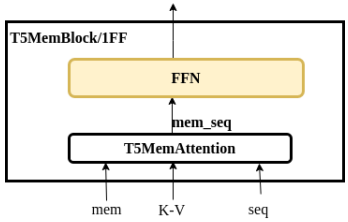

Fig. 7. 1FFN: One layer after encoder attention to process the concatenation of memory slots with their related sequence.

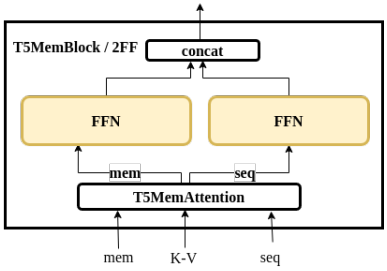

Fig. 8. 2FFN: Two layers after encoder attention, left for memory slots representations and the right to process the sequence representation resulted from the attention.

TABLE I
LOSS AND BLUE SCORES ON AGNOSTIC AWARE DATA SET.

variant	train loss	val loss	test loss	val bleu	test bleu
$T5(baseline)$	**1.455**	**1.596**	2.349	26.12	25.648
$v1$	1.466	1.632	1.634	26.34	**26.725**
$v2$	1.472	1.642	1.639	26.09	26.437
$v3$	1.465	1.629	**1.621**	**26.5**	26.706
$v4$	1.476	1.636	1.639	26.34	26.354

[a] sacreBLEU was used https://github.com/mjpost/sacrebleu

TABLE II
LOSS AND BLUE SCORES ON CONTEXT AWARE DATASET.

variant	train loss	val loss	test loss	val bleu	test bleu
$v1$	*2.93*	3.859	3.8	1.533	1.019
$v2$	*2.946*	3.86	3.837	1.944	0.882
$v3$	*2.94*	3.787	3.753	1.597	1.543
$v4$	*2.908*	3.823/	3.806	1.698	1.291

target vocabularies of 32128 tokens for all experiments. We use batch size of 160. length of all source sentences, and target is 100. Adam optimizer was used with fixed learning rate $lr = 0.00005$ with no weight decay or warm up . $d_{model} = 512, d_k = 512/8 = 64, h = 8, num\_layers = 2, d_{ff} = 2048$ If a source has one or more context then the input will be divided into the given number of sentences and rearranged in the shape shown in encoder **Fig.** 1 for each sample in the batch. In our experiments 3 context sequences were used as context. These are the settings for all experiments if it was not declared about different settings separately. For inference greedy search was used.

VII. RESULTS AND FUTURE WORK

A. Sentence level translation

Table I displays the results of the experiments using the four versions. Used data set is the context agnostic data set for sentence level translation [2]. Reference basline for translation task is the T5 transformer.

The results shows better bleu score for all versions on validation data except for v2, and better bleu scores for all versions on test data. In terms of loss still T5 transformer has better loss.

B. Context aware level translation

1) Training: Table II displays the results of training the model from scratch on the context aware data as clear model poorly performs in terms of bleu for all versions.

2) Fine tuning: Using weights of models trained on sentence level translation to train the models on context aware translation dataset we noticed that the model get better bleu scores for all versions.

C. Future work

Explore more effective design of encoder decoder attention. Pretrain the model on Masked Language Modeling task MLM,

[2]https://www.dropbox.com/s/5drjpx07541eqst/acl19_good_translation_wrong_in_context.zip?dl=0

TABLE III
LOSS AND BLUE SCORES ON CONTEXT AWARE DATASET AFTER FINETUNING.

variant	train loss	val loss	test loss	val bleu	test bleu
$v1$	*1.322*	1.678	1.665	26.12	26.615
$v2$	1.374	1.7	1.685	26.03	26.120
$v3$	*1.33*	1.698	1.667	25.88	26.356
$v4$	*1.303*	1.656	1.652	26.64	26.582

then fine-tune it on Multihop question answering task. Also this model can be used for summarization task so it will be interesting to see how the model will perform on such task.

VIII. CONCLUSION

Processing long documents has attracted a lot of researchers especially after transformers have achieved state-of-the-art results in a wide range of natural language tasks such as different sequence modeling tasks like language understanding, text summarization and translation. However, transformers are not able to process long texts due to their self-attention operation, which scales quadratically with the sequence length. Our idea is centered on processing long hierarchical documents. We started our series of experiments with translation task, to explore context aware translations.

We suggested using a transformer-based model to cope with attention limitation by adding another layer of hierarchical attention over document chunks. Specifically, we introduced the attention over memory slots related to the document chunks. Applying attention over memory slots to choose the proper chunk for the next generation step is cheaper than using the representation of the related chunks. Training the model on a context agnostic dataset then tuning the model on a context-aware dataset helped to get more precise translation and promoted translating context aware sentences.

Results showed superior bleu score for sentence level translation for all model variants compared with T5 transformer baseline, and high bleu score for context aware translation.

Future steps will be studying the results of integrated PKM memory in feed forward network layer, compare the resulted results on context aware dataset with another context aware model as baseline, pre-training this model on Masked Language Modeling task then fine tune it on other comprehension tasks such as summarization and question answering tasks.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[2] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[3] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences." in *NeurIPS*, 2020.

[4] S. Sukhbaatar, E. Grave, G. Lample, H. Jegou, and A. Joulin, "Augmenting self-attention with persistent memory," *arXiv preprint arXiv:1907.01470*, 2019.

[5] M. Cornia, M. Stefanini, L. Baraldi, and R. Cucchiara, "Meshed-memory transformer for image captioning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 578–10 587.

[6] Q. Wu, Z. Lan, J. Gu, and Z. Yu, "Memformer: The memory-augmented transformer," *arXiv preprint arXiv:2010.06891*, 2020.

[7] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-transformer," *arXiv preprint arXiv:1902.09113*, 2019.

[8] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[9] J. Ainslie, S. Ontanon, C. Alberti, V. Cvicek, Z. Fisher, P. Pham, A. Ravula, S. Sanghai, Q. Wang, and L. Yang, "Etc: Encoding long and structured inputs in transformers," *arXiv preprint arXiv:2004.08483*, 2020.

[10] M. S. Burtsev, Y. Kuratov, A. Peganov, and G. V. Sapunov, "Memory transformer," *arXiv preprint arXiv:2006.11527*, 2020.

[11] A. Gupta and J. Berant, "Gmat: Global memory augmentation for transformers," *arXiv preprint arXiv:2006.03274*, 2020.

[12] G. Lample, A. Sablayrolles, M. Ranzato, L. Denoyer, and H. Jégou, "Large memory layers with product keys," *arXiv preprint arXiv:1907.05242*, 2019.

[13] G. Kim and T.-H. Jung, "Large product key memory for pretrained language models," *arXiv preprint arXiv:2010.03881*, 2020.

[14] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[15] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[16] E. Voita, R. Sennrich, and I. Titov, "When a good translation is wrong in context: Context-aware machine translation improves on deixis, ellipsis, and lexical cohesion," *arXiv preprint arXiv:1905.05979*, 2019.

[17] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.