Dispersed Hierarchical Attention Network for Machine Translation and Language Understanding on Long Documents with Linear Complexity

Ajay Mukund S and K. S. Easwarakumar
Department of Computer Science and Engineering, Anna University, Chennai, Tamilnadu
ajaymukund1998@gmail.com, easwara@annauniv.edu

Abstract

Transformers, being the forefront of Natural Language Processing and a pioneer in the recent developments, we tweak the very fundamentals of the giant Deep Learning model in this paper. For long documents, the conventional Full Self-Attention exceeds the compute power and the memory requirement as it scales quadratically. Instead, if we use a Local Self-Attention using a sliding window, we lose the global context present in the input document which can impact the performance of the task in hand. For long documents (ranging from 500 to 16K tokens), the proposed Dispersed Hierarchical Attention component captures the local context using a sliding window and the global context using a linearly-scaled dispersion approach. This achieves O(N) linear complexity, where N is the length of the input sequence or document.

1 Introduction

Due to the advent of Transformers, the technological world, especially in the field of Natural Language Processing has grown by leaps and bounds. The impact spans from the GPT (Radford et al., 2018), (Radford et al., 2019), (Brown et al., 2020), (Black et al., 2022) models that have repeatedly made headlines in the main stream media to Github's Copilot (https://copilot.github.com). Though it is said that the Transformers model is a culmination of many ideas such as transfer learning, stacked neural networks, the basic fundamental concept that revolutionized the ideology of transformers is definitely the attention mechanism (Vaswani et al., 2017).

Let us now look at the concept of attention mechanism using a concrete example of abstractive text summarization. Earlier, the task of abstractive text summarization (Tunstall et al., 2022) was achieved using the Encoder – Decoder framework (Sutskever et al., 2014). It involves an encoder stack which encodes the entire text document to be summarized into a numerical representation called the last hidden state. Taking this as input, the summary of the input text document is given as output by the decoder. For the task at hand, it is nearly impossible for the last hidden state to contain all the important information said in the given input document. And since the last hidden state is the only piece of information that the decoder receives, the entire task in hand is compromised.

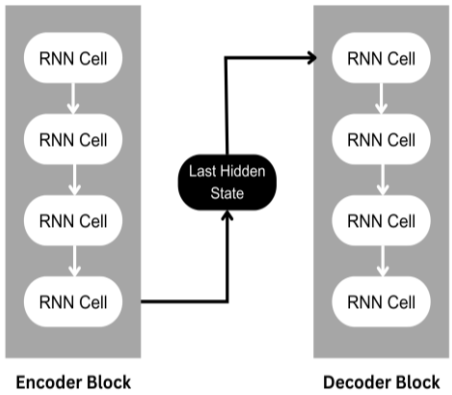

Figure 1: Encoder-Decoder Framework for a pair of RNNs demonstrating the information bottleneck.

To eradicate the shortcomings of the information bottleneck which produces sub-optimal results, at every stage, the encoder generates a hidden state, mitigating the loss of important information. In the process of overcoming the information bottleneck, we end up moving from the frying pan into the fire. It is realized that there is information overload by creating a huge input corpus for the decoder while using all the encoder-generated states simultaneously. Therefore, there is a need for a mechanism to prioritize the encoder states which must be considered and given more attention by the decoder.

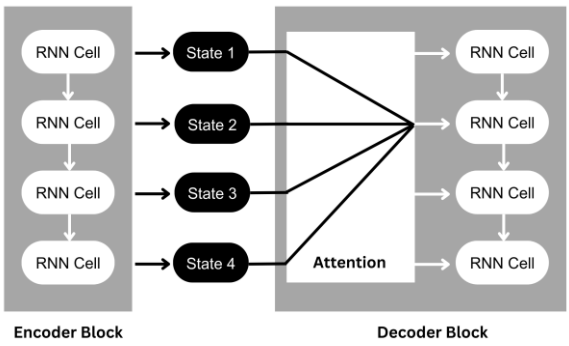

Figure 2: Encoder-Decoder framework for a pair of RNNs with an attention mechanism.

Here is a preview. This paper aims at

1) elaborating extensively on the **formulation** of different attention mechanisms
2) proposing a new **variation** of attention mechanism
3) discussing where the proposed attention mechanism can be used as an **application**
4) **evaluating** the changes in complexity which in turn leads to better performance.

2 Formulation

2.1 Self-Attention

The most predominantly used form of attention mechanism is Self-Attention (Bahdanau et al., 2014) also known as Intra-Attention. As the name suggests, instead of using a fixed encoding scheme for embedding every input token, self-attention uses the entire input document to determine the attention scores of every token present. For a particular token t and for its corresponding sequence of token embeddings $t_1,t_2,\ldots,t_n$, the results after applying self-attention creates a new sequence of embedding, $t'_1, t'_2, \ldots, t'_n$.

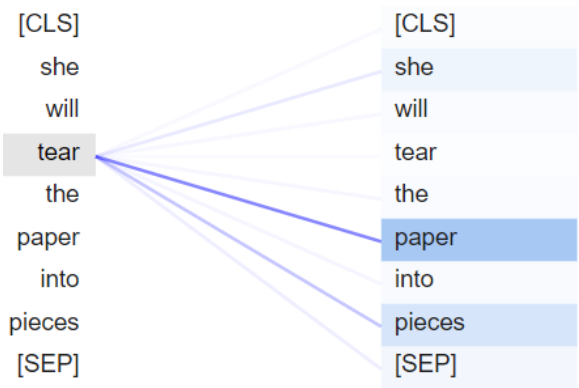

Figure 3: Based on the attention weights, the token 'tear' is giving more weightage to 'paper' and 'pieces'

Every embedding present in the new sequence (t'_i) is a linear combination of token embedding of the j^{th} token (t_j):

$$t'_i = \sum_{j=1}^{N} w_{ji}\, t_j \qquad (1)$$

t'_i s are contextualized embeddings (Peters et al., 2018) because for every token, the entire context of the document is being used via the normalized attention weights w_{ji}.

For instance, in an input sequence, "She will tear the paper into pieces" with the entire context, we can determine that 'tear' is a verb and in another sequence, "A tear rolled down her cheek", the same word is considered to be a noun with the provided contextual information. Likewise, the embedding for a token change drastically using Self-Attention which inherently takes context into account.

2.2 Scaled Dot-Product Attention

Considering a BERT (Devlin et al., 2019) transformer model, every token present in the input sequence is embedded into a vector of 768 dimensions. Therefore, if there are N unique tokens present in a sequence, the order of the entire corpus' token embedding matrix will be N x 768. Being the most common form of Self-Attention, the Scaled Dot-Product Attention (Vaswani et al., 2017) projects every single token embedding of length 768 into three different vectors namely Query (Q), Key (K) and Value (V) which are obtained via linear transformations on the input token embedding.

The similarity between Query and Key is found by simply calculating the dot-product between Q and K via matrix multiplication. Each value of the resultant vector represents how much attention must be paid towards other tokens present in the corresponding position of the vector. For an input document having N unique tokens, the resultant matrix representing the attention scores is of the order N x N. The attention scores are normalized in such a manner that it now contains the attention weights w_{ji}. Eventually, the attention weights are used on the Self-Attention formulation discussed earlier to produce the newly updated token embedding where v_j is the value vector $v_1,v_2,\ldots,v_n$ of the j^{th} token (t_j).

$$t'_i = \sum_{j=1}^{N} w_{ji}\, v_j \qquad (2)$$

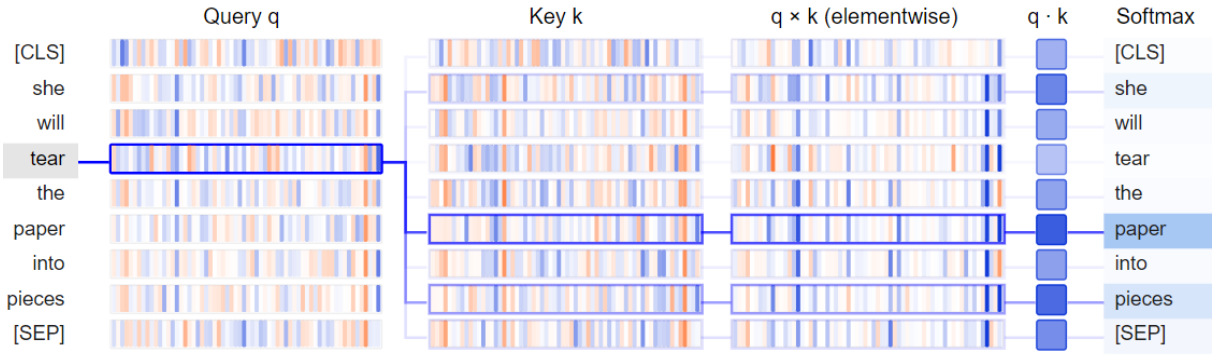

Figure 4: Visualization of computation of attention weights using Query (q) and Key (k) vectors

The visualization in Figure 4 helps in picturizing the entire process of Scaled Dot-Product Attention through the neuron_view module present in the BertViz Library (Vig, J., 2019).

2.3 Multi-Headed Attention

As discussed in section 2.2, Query (Q), Key (K) and Value (V) vectors are a result of the independent linear transformations acting on the input sequence token embeddings. The attention head, formed by the three vectors, is responsible for the resultant attention scores. In Multi-Headed Attention, the existing attention mechanism is improvised by using multiple attention heads and parallelizing the computation to form its corresponding Scaled Dot-Product Attention.

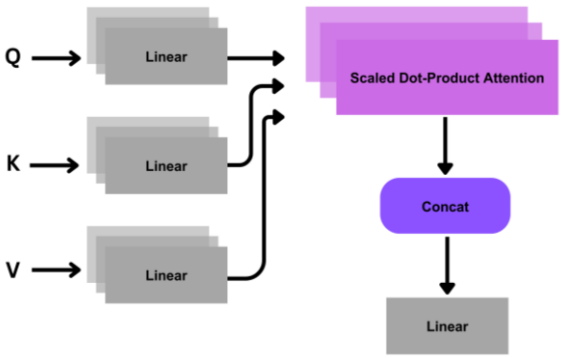

Figure 5: Flow of the Multi-Headed Attention Layer

Why is there a need to use multiple heads? These multiple heads can be compared to the numerous filters present in a typical Convolutional Neural Network (CNN) (Lecun et al., 1998) where each and every filter extracts a set of features from the given image. Likewise, in a Natural Language Processing setting, each and every attention head can be used to extract different dependencies and correlations from an input sequence ranging from evaluating the subject-verb agreement to finding articles, adverbs, adjectives, prepositions, conjunctions and interjections.

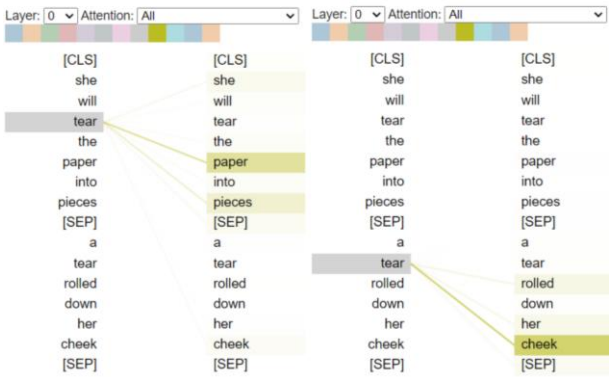

Figure 6: Identification of 'tear' as a verb and a noun based on context and its corresponding dependencies via Multi-Headed Attention

2.4 Sliding Window Attention

Matrix multiplication being a very compute intensive task, finding the attention scores for all the tokens present in a long document having a very large corpus of length, say N, will result in an N x N matrix. To reduce the complexity of this operation and to take the local context instead of the global context into consideration, the Sliding Window Attention (Luong et al., 2015), also known as Local Self Attention, was proposed.

In Sliding Window Attention, a window of fixed length, w is considered and the attention scores are calculated only for the neighboring tokens of any particular token. Exactly, for w/2 number of neighboring tokens on either side of the current token is to be taken as part of the local window. Through this, the complexity of the original attention which is $O(N^2)$ being quadratic, is reduced to O(N), thus becoming linear. This is achieved as there will only be N x w non-zero attention scores in the N x N attention matrix.

2.5 Dilated Sliding Window Attention

Inspired by the concept of Dilated CNNs (Lei et al., 2019), a new attention mechanism was brought forth which came to be known as the Dilated Sliding Window Attention. It is computationally equivalent to the Sliding Window Attention discussed earlier but it has a larger receptive field. This difference is achieved by using fixed evenly size gaps of length d in between each neighboring token. If in a case, the fixed window size, w is 4 and the fixed dilation size, d is 1, Figure 7 pictorially represents the set of neighboring tokens that are considered for the calculation of attention scores with the entire input sequence, N spanning to 20 tokens.

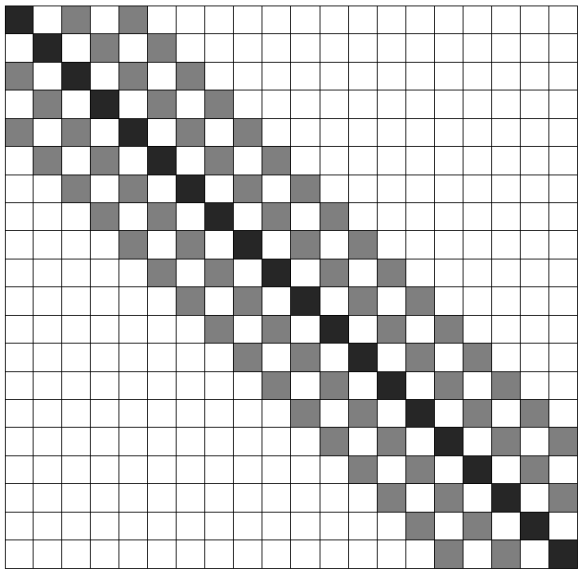

Figure 7: Dilated Sliding Window Attention containing number of tokens, N = 20 with window size, w = 4 and dilation, d = 1

In the context of a Multi-Headed Attention mechanism, Beltagy et al., 2020 discusses that the performance of the transformer model improves while using a conventional Sliding window attention (without dilation) to capture the local context present in the input text document. On the other hand, in a different attention head, using a Dilated Sliding Window Attention mechanism with varying configurations help to capture the important tokens on a context much larger than the conventional one.

3 Variation

From the plethora of attention mechanisms (Niu et al., 2021) that have been explained in the previous sections, we find a commonality. If an attention mechanism tries to comprehend the entire input document using the Full (Global) Self-Attention, the computational requirements and the memory requirements increase in a quadratic manner. While processing long documents, the above stated methodology is rendered incompatible. Acting upon this, if we downsize the attention mechanism to Sliding Window or Dilated Sliding Window, though the complexity and the memory usage come down drastically, the global context of the entire input document is lost and only a local context is captured.

Here is a variation in the attention component which tries to comprehend the global context of the entire document along with the local context. Intuitively, the neighboring tokens contain more weightage for any given token and therefore ideally these said tokens have to be given more attention. As we move away from the token, the importance of the respective tokens which are not in close proximity decreases. This intuition forms the theoretical basis of the proposed attention component – **Dispersed Hierarchical Attention**. Dispersed Hierarchical Attention is a combination of Sliding Window Attention along with a linearly-scaled dispersion on either sides of the token embedding.

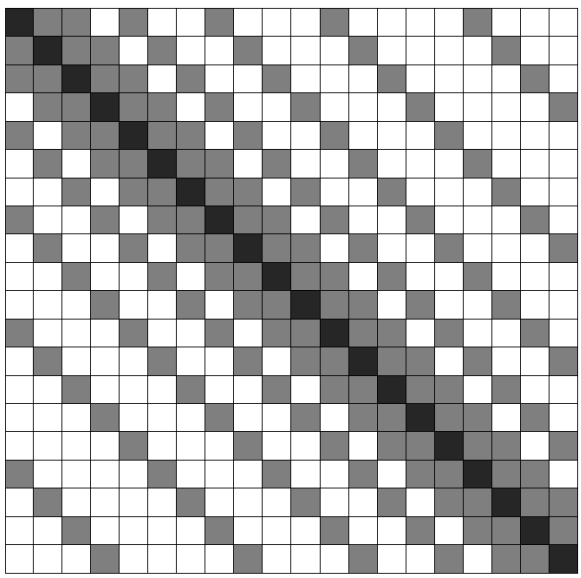

Figure 8: Dispersed Hierarchical Attention containing number of tokens, N = 20 with window size, w = 4

The sliding window contributes to the local context and the linearly-scaled dispersion considers token with reducing importance with respect to the decrease in proximity. This methodology still manages to capture the global context to some extent as well. From Figure 8, though it may not be evident for a small scope of N = 20 tokens, the dispersion occurs in a linear manner as we lose proximity with the corresponding token embedding thus reducing the quadratic complexity.

$$\text{Att}_{\text{Window}} = N + (N - w)w + 2\sum_{i=1}^{w/2}(w - i) \quad (3)$$

Equation (3) calculates the total number of tokens for which the attention scores are calculated while using the Sliding Window Attention (Figure 9) of size, w and the entire sequence length, N.

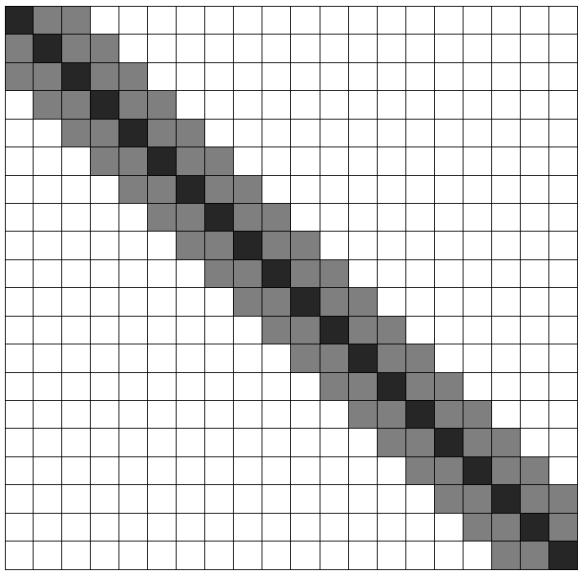

Figure 9: Sliding Window Attention containing number of tokens, N = 20 with window size, w = 4

Algorithm 1 – To find the calculated number of Attention Scores excluding the Sliding Window

---

Input:

1. **N** (integer) – Number of Tokens
2. **w** (integer) – Size of the Sliding Window

Output:

Total_Attention (integer) – Number of scores to be calculated

Description: This algorithm calculates the total number of attention scores that will be taken into account while using the Dispersed Hierarchical Attention Component excluding the tokens that are a part of the defined sliding window.

Step 1: Read the input values **N** and **w**
Step 2: Initialize variables
- a) **count** to 3
- b) **num** to 1
- c) **i** to 0
- d) **sum** to 0
- e) **flag** to True

Step 3: While **flag** is True, repeat the following:
- a) for each **j** in the range of **count**
  - (1) If **i** is equal to **N - w**, set **flag** to False and exit the loop
  - (2) Increment **i** by 1
  - (3) Add **num** to **sum**
- b) Increment **num** by 1
- c) Increment **count** by 1

Step 4: Calculate the **Total_Attention** as **2*sum**
Step 5: Output (or) return the **Total_Attention**

Taking the values from the figure shown above, we calculate the total number of attention scores to be calculated from the attention matrix by substituting the values in Equation (3) and the total number of attention score is found be 94. The total number of attention scores while using the Dispersed Hierarchical Attention can be found by adding the number of attention scores present in the sliding window from equation (3) and the number of attention scores that are given as output from Algorithm 1.

We can also mathematically find the number of attention scores via the steps given below. If you notice the attention matrix of the Dispersed Hierarchical Attention mechanism, there is an innate series of number of tokens to either side of the token excluding the sliding window.

The series is as follows:
1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, …

$$x = \text{floor}\left(\left(\sqrt{2(N - w + 3) + \frac{1}{4}}\right) - \frac{1}{2}\right) - 2 \quad (4)$$

Equation (4) will give us the value of x which denotes till which number the above series will go on given the sequence length, N and window size, w. Basically, all the numbers in the series must be summed up to find the number of attention scores to be calculated on either the lower triangular matrix or the upper triangular matrix of the Attention matrix.

$1 + 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 + \ldots$
Let the x^{th} term in the series be T_x and let S_x be the sum of all the terms present in the series, then

$$S_x = 3 + 8 + 15 + 24 + \cdots + T_x \quad (5)$$

Since $T_1 = 1(1+2) = 3$, $T_2 = 2(2+2)$, $T_3 = 3(3+2)$
We derive that,

$$T_x = x(x + 2) \quad (6)$$

Substituting (6) in (5), we get

$$S_x = \sum T_x \quad (7)$$

Solving (7),

$$S_x = \frac{x(x+1)(2x+7)}{6} \quad (8)$$

One important point that has to be noted is that, based on the number of input tokens, N, there is a possibility that the series is never fully complete and therefore in that case a part of the last entry has to be removed and for that we do the further calculations given below.

$$\text{Remainder} = ((N - w) - \sum_{m=3}^{d+2} m)\,(d - 1) \quad (9)$$

The remainder is a (negative) number that can be subtracted from S_x. Therefore, the final refined formula for finding the total number of attention scores, $Att_{Dispersed}$ is given below via Equation (10) by combining the Equations (3), (8) and (9)

$$Att_{Dispersed} = Att_{Window} + S_x + \text{Remainder} \quad (10)$$

4 Application

In the previous section, the proposed variation for the attention component present in the transformer model was discussed. It also provided us with the formulation of the total number of attention scores for any given sequence of input length, N and window size, w. The next step is to create a function (Algorithm 2) for the Dispersed Hierarchical Attention Mechanism so that it can be replaced with the existing attention functions and can be imbibed in a transformer.

The algorithm divides the sequence into local windows (sliding window) and calculates the attention scores within those windows to capture the local context. To capture the global context, attention scores for the dispersed tokens to the right and left of the local window are calculated. The gap sequence defined in Algorithm 2 spans from 2 to 180 in order to accommodate the input sequence length up to 16470 tokens (Equation (11)).

$$\frac{n(n+1)}{2} + n \quad (11)$$

Since, the Longformer (Beltagy et al., 2020) Transformer can take documents containing tokens up to 16384, the specified gap sequence is the maximum that the proposed attention can span.

Algorithm 2 – Dispersed Hierarchical Attention

Input:

1. **sequence** (2D array) – A sequence of input tokens where each token is a 1D array (token embedding).
2. **window_size** (integer) – The size of the sliding window.

Output:

attention_matrix (2D array) – The matrix representing the attention scores between tokens in the input sequence.

Description: The "Dispersed Hierarchical Attention" algorithm calculates attention scores between tokens present in the input sequence using a linearly-scaled dispersion approach.

Step 1: Initialize variables

a) **sequence_len** to length of the sequence
b) Create an empty 2D array **attention_matrix** of size sequence_len x sequence_len and initialize all the elements to zero
c) **window_size** to window_size integer divided by 2

Step 2: For each token in the sequence, repeat the following:

a) Initialize a loop for **size** in the range of **window_size + 1**
   i) Calculate the right index **j** as **i + size**
   ii) If **j** is less than **sequence_len**, attention_matrix[i, j] is the attention score calculated between token in index i (current token) and j
   iii) Calculate the reverse index **rev** as **i – size**
   iv) If **rev** is greater than or equal to **0**, attention_matrix[i, rev] is the attention score calculated between the current token and the token present in index rev
b) Create a list **gap** representing a gap sequence **[2, 3, 4, …, 180]**
c) For each gap **g**, update **j** as **j + g**

i) If j is less than sequence_len, attention_matrix[i, j] is the attention score calculated between token in index i (current token) and j

d) Update **j** as **i – window_size**

e) For each gap **g**, update **j** as **j - g**

i) If **j** is greater than or equal to **0**, attention_matrix[i, j] is the attention score calculated between token in index i (current token) and j

Step 3: Return the **attention_matrix** containing attention scores between tokens

Once the tokens are identified, say i and j, a range of similarity metrics such as a simple dot product of the 1D vectors, cosine similarity (Graves et al., 2014), Euclidean distance, Manhattan distance or using more complex calculations like a custom similarity metric (Thongtan et al., 2019) can be used to calculate the actual attention score between token i and token j.

5 Evaluation

To evaluate the performance of the proposed attention, we use a comparative approach (Table 1) with differing lengths of input sequences on Full (Global) Attention, Sliding Window Attention and Dispersed Hierarchical Attention. From the second column in Table 1, it is evident that, as the input sequence N increases, there is a massive surge in the number of attention scores computed which is the entire attention matrix while using the Full Self-Attention exhibiting quadratic scaling.

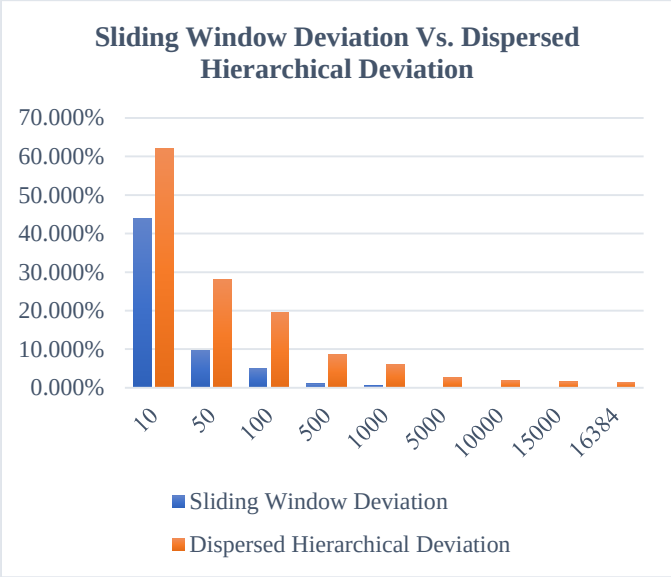

Figure 10: Difference between the above two quantities is the percent of Attention scores calculated to either sides of the Sliding Window

From columns 3 and 4, it is observed that the Sliding Window Attention substantially decreases the aforementioned requirements. When the document length, N becomes greater than 500, the attention matrix is a very sparse matrix with less than 1% of the entire elements being non-zero. The sliding window does its role of reducing the complexity but it comes with the cost of not capturing the global context which plays a significant role in Natural Language Processing and Natural Language Understanding tasks such as Text Summarization and Sentiment Analysis.

Sequence Length / # Tokens in the input document, N	**Window Size, w = 4**				
	Full Attention $O(N^2)$	**Sliding Window Attention O(N) – Local Context**		**Dispersed Hierarchical Attention O(N) – Global + Local Context**	
	Total No. of Calculated Att. Scores	**Total No. of Calculated Att. Scores**	**Deviation from Global Self Attention**	**Total No. of Calculated Att. Scores**	**Deviation from Global Self Attention**
10	100	44	44.000%	62	62.000%
50	2500	244	9.760%	700	28.000%
100	10000	494	4.940%	1962	19.620%
500	250000	2494	0.998%	21524	8.610%
1000	1000000	4994	0.499%	60550	6.055%
5000	25000000	24994	0.100%	671500	2.686%
10000	100000000	49994	0.050%	1895358	1.895%
15000	225000000	74994	0.033%	3478806	1.546%
16384	268435456	81914	0.031%	3970510	1.479%

Table 1: Comparative Analysis of Full, Sliding Window and Dispersed Hierarchical Attention with window size of 4 and with varying sequence length, N

Even if the sliding window size w is varied, the results don't vary substantially in terms of deviation and therefore, the window size is normalized at w = 4. The proposed mechanism – Dispersed Hierarchical Attention – does a fair job of capturing the local and the global context. During which, it keeps the number of attention scores calculated in the resultant sparse attention matrix to a bare minimum. In the process, the newly defined attention component achieves linear complexity. On comparison, the deviation of Sliding Window and Dispersed Hierarchical Attention from the Full Self-Attention (Figure 10), it is visualized that even for very large documents for which the sequence length, N is up to 16384 tokens, the Dispersed Self Attention gives approximately 1.5% of the entire attention matrix (~39 Lakh attention scores) that will intuitively give an overall picture of the entire document instead of relying on the Sliding Window Attention which gives a meager 0.03% of the attention matrix (~82K attention scores) that captures the local context.

6 Conclusion and Future Work

This novel work mainly focuses on reducing the complexity of the attention mechanism used in the transformer model. In the proposed attention mechanism, selective tokens which are linearly dispersed from the query are used as keys. This addresses the problem of considering the local and global context associated with large input documents while potentially lowering computing and memory costs. The solution proposed brings the understanding of both the local and global context to middle ground. In future, the proposed Dispersed Hierarchical Attention mechanism can be integrated with a series of transformer models replacing the existing attention mechanisms implemented and could be tested on the efficiency improved in performing various NLP tasks such as Text Summarization, Question Answering, Sentiment Analysis and Machine Translation.

7 References

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI blog 1, no. 8 (2019): 9.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. Advances in neural information processing systems, 33, pp.1877-1901.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. arXiv preprint arXiv:2204.06745.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in neural information processing systems, 30.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27.

Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2022. Natural Language Processing with Transformers. United States: O'Reilly Media.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short

Papers), pages 4171–4186, Minneapolis, Minnesota.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. arXiv preprint arXiv:1906.05714. 12 Jun 2019. https://doi.org/10.48550/arXiv.1906.05714

Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025. https://doi.org/10.48550/arXiv.1508.04025

Xinyu Lei, Hongguang Pan, and Xiangdong Huang. 2019. A dilated CNN model for image classification. IEEE Access, 7, pp.124087-124095.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150. https://doi.org/10.48550/arXiv.2004.05150

Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. 2021. A review on the attention mechanism of deep learning. Neurocomputing, 452, pp.48-62. https://doi.org/10.1016/j.neucom.2021.03.091

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. arXiv preprint arXiv:1410.5401. https://doi.org/10.48550/arXiv.1410.5401

Tan Thongtan and Tanasanee Phienthrakul. 2019. Sentiment Classification Using Document Embeddings Trained with Cosine Similarity. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop, pages 407–414, Florence, Italy. Association for Computational Linguistics.