A Kernel based approach to Learning Sparse Attention in Transformers

Deep Karkhanis[*], **Manley Roberts**[*], **Soundarya Krishnan**[*]
Machine Learning Department
{dkarkhan, mhrobert, soundark}@cs.cmu.edu
Carnegie Mellon University, Pittsburgh, PA - 15213

1 Introduction

Transformers are wildly successful deep models popular in the fields of natural language processing [1], vision [2], and multimodal learning [3]. One obstacle when using these models is the massive size of transformers such as BERT [1] and RoBERTa [4]–which requires practitioners to allocate massive chunks of memory for model storage and schedule significant compute time for inference and training on large corpora.

One significant computational bottleneck in transformers is the quadratic complexity that occurs in the attention mechanism of each transformer block. For each head in a transformer block, (scaled dot product) attention is calculated as $attention(Q, K, V) = \pi(\frac{QK^\top}{\sqrt{d}})V$. $Q \in \mathbb{R}^{n\times d}$ is the d dimensional representation of n queries, $K \in \mathbb{R}^{m\times d}$ is the d dimensional representation of m queries, and $V \in \mathbb{R}^{m\times d}$ is the d dimensional representation of m values. π is any function that maps vectors to distributions, typically softmax. The main computational barrier is in the matrix product, $QK^\top$, which takes $O(nmd)$ time. m and n depend on the sequence length, and thus, transformers are quadratic in the sequence length, which leads to slow training and inference on long sequences found in tasks like abstract summarization or genomic data representation extraction.

Several methods seek to decrease the computation associated with the $QK^\top$ product by limiting the number of other inputs in a sequence to which a model attends when calculating the attention layer output for a given input. These methods are part of a category of computationally improved transformers we call efficient transformers.

Several prior works detail efficient transformers (see Section 3). We posit a different approach towards generating sparse attention patterns that involves the use of 1D Kernels to learn the sparse attention for every row in the $QK^\top$ matrix. We train a module to predict a sparsity mapping (top-k softmax outputs over $QK^\top$) across the sequence, given a small window of token embeddings around the target token. This prediction is done with the use of Kernels (either fixed or learned), and the goal is to predict a probability of each kernel being needed to produce the final sparse output, while at the same time potentially learning these kernels themselves.

Through this approach, our goals are twofold: (a) computational, which we measure through inference clock time, memory at inference, and FLOPS required at inference time, and (b) performance, which we evaluate by measuring validation perplexity on a Masked Language Modeling task. Note that at present, we are primarily interested in inference-time costs as opposed to training-time costs, and we hope to eventually demonstrate gains during train time as well.

2 Background

In our midway report, we examined existing sparse transformer approaches, particularly LongFormer [5] and EntMaxer (a form of Roberta with the entmax sparse approximation to softmax, based on

[*]Equal contribution, names listed in alphabetical order

the work of [6]). Longformer, employing fixed limited attention patterns, observed fast inference and good performance. EntMaxer observed slower inference (due to an inefficient implementation which still included computing all attention values, even those that would be sparsified), and worse performance (due to some incompatibility with weights pretrained on softmax). We used the success and efficiency of [5] and other fixed-attention-pattern sparse approaches such as [7] when designing our kernel-based approach.

3 Related Work

We refer to [8], which presents a survey of key practices, with a varying degree of complexity, for sparsified transformers. We discuss a few relevant ones here:

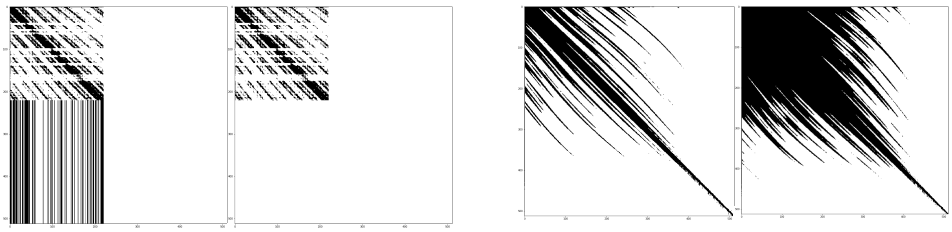
Fixed Patterns: This approach involves utilising underlying intrinsic patterns to approximate full attention. Such patterns involve diagonal sliding window (local) patterns that are sometimes strided or dilated, as well as vertical or horizontal (global) patterns. Sparse Transformer [9], which uses a combination of local and strided patterns, Longformer [5], which uses dilated sliding windows with a global task-specific attention, and BigBird [7], which uses a combination of local, global, and random patterns, are examples of such an approach.

Learning Attention from Data: This approach involves learning which elements are non-zero in the squared attention matrix so as to avoid computing them. Reformer [10] is one example of such an approach in which a hash-based similarity measure is used to compute only those terms in the attention matrix after the softmax operation that are significant. Sparsefinder [6] utilises a sparse version of softmax, α-entmax [11], and subsequently predicts which elements in in the $QK^\top$ matrix will be 0. Given B distinct buckets, their approach focuses on predicting mappings from keys and queries to 2^B, meaning that each key or query will be assigned to some subset of the the buckets. The attention head will only calculate the attention between a key and query if they share some bucket.

We wish to combine the two approaches listed above, i.e. to learn attention *patterns* from *data*. The key idea is to exploit the efficiency of fixed patterns, while at the same time, utilising the task-specific, and dataset-specific aspects of the transformer. As discussed, several previous approaches achieve success by combining a fixed local attention window with some fixed, random, or custom task-specific global attention pattern. Our first novel contribution is that we perform more intelligent unified estimation of the global and local attention patterns in a given attention matrix. Our approach is a generalisation of previous approaches which learn attention patterns from data, in which we don't assume a similarity function (cosine-similarity, euclidean distance, etc.) which ranks the keys that queries should attend to, and instead learn it directly using a neural network. We also find that several works (e.g. [7], [5]) assign a specific pattern to all attention blocks across heads and layers. We believe that this is a flawed approach after analysing the attention patterns in different heads and layers, and find that different heads and different layers are learning different aspects of data which are reflected in the patterns present, as analysed in Sec. 4. Our second novel contribution is that we account for this diversity by utilising a different model for each head and layer.

4 Methods

As is common in sparse transformer literature [6], our method starts with a large pretrained dense transformer, `roberta-base` [4]. We use the WikiText-103 dataset [12], which makes use of over 100 million tokens of text from Wikipedia (specifically, articles that are Good or Featured). For certain experiments, we use the WikiText-2 dataset (a subset of WikiText-103) to account for lack of computational resources. We have also created a $1/100^{th}$ size subset of WikiText-2 which we call WikiText-0.02 for the hardware performance analysis of our models. We do a forward pass of a dataset on our model and use it for the Masked Language Modeling (MLM) task, in which a partially ablated sentence or text chunk is fed into the transformer, and the expected output is the original sentence (e.g. the model must learn to fill in missing tokens or words based on context). We use a masking probability of 15%, and pad all sequences to a multiple of 512. We run all experiments on a machine that consists of 256 GiB RAM, 48 CPUs, and 8 GPUs (4 having 10 GiB RAM and 4 having 24 GiB RAM). We use batch size of 8 in our forward passes.



(a) Before and After Removal of Padding Artifacts

(b) Accumulated attentions before and after length normalization

Figure 1: Processing attention maps

4.1 Analysing RoBERTa Attention Patterns

We obtain input tokens and attention probabilities for each head and each layer from a hook in the forward pass of RoBERTa. We then scale these attention probabilities based on the frequency of occurrence of each sequence length. This removes sequence padding artifacts from the matrix (see Fig. 1(a). Further, since a matrix doesn't have a uniform distribution of element frequencies around its diagonal (there are more elements closer to the diagonal than further away), we scale the attention probabilities for relative position from the principal diagonal before generating local kernels (see Fig. 1(b)). We visualize the top k=20 values in each row, and find that the patterns in different heads and layers are diverse (see Fig. 3). This builds the case to have different models for every head and layer.

4.2 Kernel-based approach and Experimental Setup

We recall that typically the attention module does not output sparse outputs. However, as the attention from each token across the sequence is softmaxed, we often observe a few tokens strongly attended to by each token, and many tokens very weakly attended to. We convert this near-sparsity to full sparsity by binarizing attention probability values; those below a threshold of 0.05 are sent to 0, and those above are sent to 1.

The goal of our attention prediction module is to replicate the exact sparsity pattern as this thresholded softmax, after seeing only a very small window around each token. This model must be lightweight to avoid creating another source of quadratic complexity (or worse). We thus propose a model that learns the probability of occurrence of certain kernels, which are either standard, or handcrafted from the dataset. We also propose to simultaneously learn a small collection of distinct pattern kernels along with the probabilities of their occurrence. At inference time, for every row in the attention matrix, we propose to threshold these probabilities to select the kernels - which, when OR'ed together, will produce the set of tokens that will be attended to. We will then stack the OR'ed patterns for every row to produce the final attention pattern 2.

For training the 1D Kernel model, we train a model $f_\theta(x)$ with the input x as the local window around the token, and the output y as the predicted top-k values in $softmax(Q_i K^\top)$, where Q_i is the query vector of that token. $\hat{y}$ is generated with the help of Kernels κ_i's and an associated probability α_i that is learnt. We take a local window of the input tokens and top-k attention probabilities from the hook function mentioned in the previous section as the training inputs for this model.

For our kernel prediction module, we define a separate predictor for every layer. Each predictor consists of a combination of linear layers and ReLU and Sigmoid activation functions. We compare the performance with 2 and 3 hidden layers, and a hidden size of 32 or 64 for each hidden layer. We send the embeddings of tokens in a small local window around the query token (before it is transformed into the query vector) along with a location embedding. The size of the kernel is two times the size of the sequence length, and it gets truncated according to the sequence length, and the amount by which the kernel hangs off the attention matrix. In our implementation, the input embedding size is 768, and the location embedding size is 4. The location embedding helps the model identify where in the sentence the token is present.

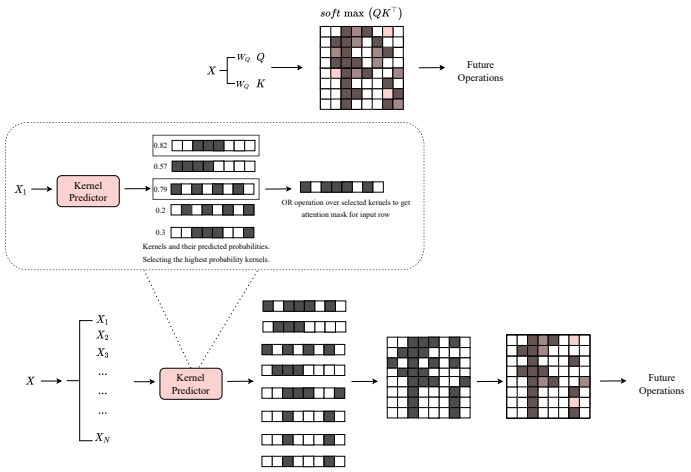


Figure 2: End-to-end pipeline of kernel-based approach for sparsifying attention matrix

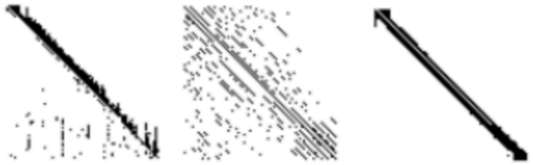


Figure 3: Diversity of attention maps in different heads and layers

We compare three main approaches in our investigation: standard kernels, handcrafted (from data) kernels, and learnt kernels.

4.2.1 Standard Kernels

As discussed in Sec 3, past works have found sliding window and dilated patterns to be useful, along with global patterns. Most of the previous work on fixed pattern sparse transformers can be reduced to making use of different sets of attention matrix patterns. We compile these sets of patterns into 1D-Kernels and train our attention prediction module on them. This essentially creates an ensemble model combining most prior work on sparse transformers.

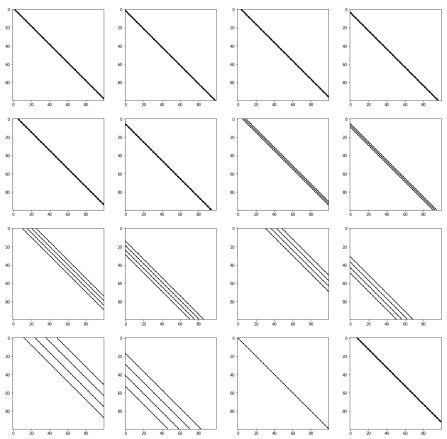
We compile 17 kernels: 8 of them focus on attending to tokens after the current token, another 8 focus on attending to previous tokens, and one identity kernel which attends the current token to itself. The kernels other than the identity token vary in the length of their sliding window and the amount of their dilation as shown in Fig. 4(a) (a 100×100 figure is shown for clarity purposes).

4.2.2 Handcrafted Kernels

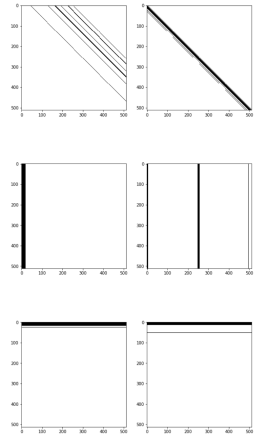
In the following experiments, we approximate the attention matrix with dataset-based attention patterns. We handcraft kernels based on statistical analysis of the training data. We consider this an intermediate step to having fully trainable kernels. For each head and layer, we define 2 new local kernels and 4 new global kernels. The global kernels are of two types: (i) *Global Rows*: Queries which attend to every Key, (ii) *Global Columns*: Keys which are attended by every Query

For all 12 heads and 12 layers, we extract the set of 2D attention matrices by performing a forward pass of WikiText-2 training data on RoBERTa. We then scale these attention probabilities based on the frequency of occurrence of each sequence length. This removes sequence padding artifacts from the matrix.

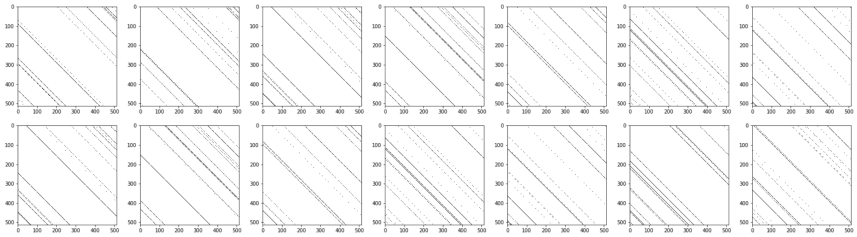
To extract 1D-Kernels from this set of accumulated 2D attention matrices, we perform assimilation operations. Each operation gives a new kernel and is meant to capture a unique pattern in the data:



(a) Standard Kernels



(b) Handcrafted Kernels



(c) Learned Kernels

Figure 4: Kernel visualizations

Top-k pruning: We sum up the probabilities for each of the dimensions along a specific axis and pick the top-k indices. We set the kernel to be one only in these k indices and 0 otherwise. To ensure sparse kernels, we set k=20 for our experiments.

Correlation based selection: Along the specified axis, we pick the index which has the highest average attention probability. We then choose k=20 indices whose attention probabilities along this axis are most correlated with this chosen index. We ensure that the original most probable index is present in this choice since its correlation with itself will be one and a 20-fold tie-breaking scenario is unlikely. The correlation based kernel will have exactly these indices set to one. Since picking a kernel will trigger the attention computation of all its 1-indices together, we intuit that the ones in a kernel should occur at indices which are closely correlated in their attention sparsity pattern.

Performing these assimilations along the row and column of the attention matrix gives the global column and global row kernels respectively. In case of local kernels, we perform the assimilation along the principal diagonal of the matrix. Since a matrix doesn't have a uniform distribution of element frequencies around its diagonal (there are more elements closer to the diagonal than further away), we scale the attention probabilities for relative position from the principal diagonal before generating local kernels. Fig. 4(b) shows the handcrafted kernels for the 1st layer and 1st head.

4.2.3 Learnable Kernels

In our next set of experiments, we made the kernels themselves to be trainable and learned them along with their coefficients. All kernels used here are local kernels. Unlike the discrete binary representation in previous experiments, the kernels here are a vector of real numbers passed through a sigmoid non-linearity. This continuous representation allows for kernel training while ensuring their values stay between 0 and 1. We use 14 randomly initialized 1023 dimensions long local kernels for each layer. See Fig. 4(c) for learnt kernels for the 1st layer and 1st head.

4.3 Loss Function and Evaluation Metrics

4.3.1 Attention Prediction Loss Function

We define the loss $\mathcal{L}$ as a function of the attention prediction module's parameters θ:

$$\mathcal{L}(\theta) = \sum_{s} \left(\texttt{max}(\alpha_i(\theta)\mathcal{K}_{is}) - a_s\right)^2 \tag{1}$$

We utilise a max operation as a max operation over binary variables is equivalent to an OR operation. Thus, the loss function is optimized when the OR(.) over all the chosen kernels is as close to the actual attention mask $[a_1, a_2, ..., a_n]^T$ as possible. Note here that our OR(.) is over binary variables $\mathcal{K}_{is}$ scaled by the importance of their respective kernels α_i.

4.3.2 Evaluation Metrics

In support of goal (a), we obtain computational costs for each baseline along several dimensions. As stated in [13], no one single metric is sufficient to reflect computational cost, and some of these metrics contradict each other. We report FLOPs at train time, expected inference time FLOPs, peak GPU usage, training time per epoch, and number of model parameters. In support of goal (b), we perform a Masked Language Modeling experiment on validation data, and report final task perplexity while varying the sparsity threshold and degree of sparsification. We report sparsity as the average number of "1"s in the attention mask in a head. It is averaged across all heads and layers

5 Results

5.1 Training and Validation on WikiText data

Table 1: Comparing our models on WikiText-2 and 200,000 training points of WikiText-103

	WikiText-103			WikiText-2		
	Train Set	Validation Set		Train Set	Validation Set	
	Loss	Perplexity	Sparsity	Loss	Perplexity	Sparsity
Standard kernels	23652.53	3419.97	856.88	**285984.66**	**2487.25**	669.54
Handcrafted kernels	29432.47	3448.61	**68.39**	313981.66	2677.21	645.40
Learned kernels	**21341.62**	**3413.06**	1743.73	330944.50	2631.72	**542.02**

Table 2: Compute Requirements for different kernel approaches during training on WikiText-0.02. FLOPs and Model Parameter are truncated to order of magnitude, however, bolded values indicate best values prior to truncation.

	FLOPS	Peak GPU usage	Time/epoch	Model Parameters
RoBERTa	-	-	-	1.253×10^8
LongFormer	6.321×10^{11}	42.31 GiB	228s	1.493×10^8
EntMaxer	5.265×10^{11}	38.14 GiB	190s	1.247×10^8
Standard kernels (ours)	2.561×10^{11}	23.11 GiB	76s	1.247×10^8
Handcrafted kernels (ours)	$\mathbf{2.561 \times 10^{11}}$	**22.97 GiB**	**74s**	$\mathbf{1.247 \times 10^8}$
Learned kernels (ours)	2.561×10^{11}	23.65 GiB	77s	1.247×10^8

Table 3: Inference FLOPS on WikiText-0.02. Values are theoretical estimates calculated from empirical sparsity and number of model parameters

	Inference FLOPS	Reduction in FLOPS
Dense Transformer	2.902×10^{10}	-
Standard kernels (ours)	2.431×10^{10}	16.23%
Handcrafted kernels (ours)	$\mathbf{2.278 \times 10^{10}}$	**21.50%**
Learned kernels (ours)	2.462×10^{10}	15.16%

We trained three classes of models based on the different types of kernel definitions given in 4.2. Table 1 compares the models for their training loss, and their perplexity and sparsity on the validation

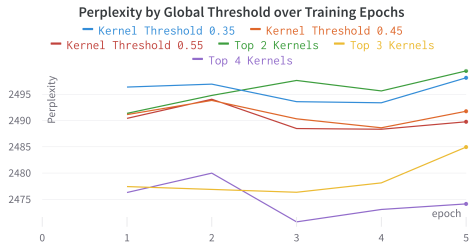


Figure 5: Final Masked Language Task Perplexity of sparsified model plotted over epochs of training on Wikitext-2.

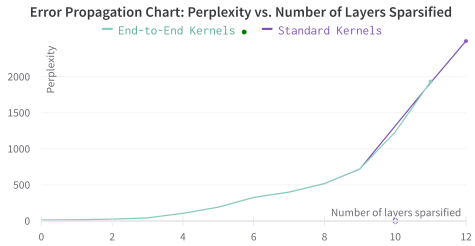


Figure 6: Error Propagation Plots.

set. We compare on the WikiText-2 and Wikitext-103 dataset. In case of WikiText-2, the standard kernels get the best validation perplexity and training loss, while learned kernels seem to have the best validation sparsity. This can be attributed to the fact that the standard kernel model is much easier to train even with a small dataset like WikiText-2. When a larger dataset i.e WikiText-103 is used, the barrier for learning is crossed and learnable kernels seem to have the best performance in training loss and validation perplexity. However, their sparsity gains are lost. This implies the kernels do not directly learn sparsity from data either because the attention inherently isn't sparse or the learning of sparsity needs to be more explicit. We aim to fix this by adding a explicit sparsity term to the loss in our future work. Handcrafted kernels get the best validation sparsity indicating that even simple kernels designed from correlation or top-k assimilations fit the data very well and sparsify them easily.

We also compared hardware performance metrics for end to end runs of our models as sparse transformers in Table 2. We compare them with the dense transformer (RoBERTa) baseline and other state-of-the-art sparse transformers. These experiments are done on a $1/100^{th}$ size of WikiText-2 which we call WikiText-0.02. Since our model training involves only training the kernel predictor module and no backward pass on the entire RoBERTa model, our training times, GPU usage and FLOP counts are much lesser than other models even though the total number of model parameters remain similar. The handcrafted kernels model performs the best on all these metrics. This is mainly attributed to the model having the least number of kernels thus having the least number of computational operations. Since the model also performs better modeling of training data owing to the correlation analysis, the handcrafted kernel model upon more training can potentially be exploited for constructing a new-state-of-the-art transformer.

In Table 3, we show the theoretical estimates of potential FLOP reductions that our models can achieve, They are calculated from empirical values of the sparsity metric and number of model parameters. Owing to lesser number of kernels, the handcrafted kernels model shows the best reduction in FLOP count. This number is higher than any other sparse transformer model we have witnessed on this dataset.

5.2 Masked Language Modeling Task Performance

For evaluation, we apply our learning masking model to attention probabilities calculated at each layer. We pass intermediary activations to the model, predict coefficient outputs for each kernel, and then select a subset of kernels either by hard thresholding kernel coefficients (and choosing those above threshold) or choosing a top-k kernels. At every layer, attention masks are produced either by selecting only kernels with coefficient above a fixed threshold, or choosing the top K kernels. Kernels are standard kernels. Results are available in Fig. 5. As we can see, perplexity is roughly similar for all thresholds, but choosing larger top k (and therefore selecting more kernels) produces a more expressive kernel map and leads to improved (lower) perplexity.

5.3 Error Propagation Analysis

Other perplexity plots in this report assume that we use our kernel sparsified approach in an "all-or-nothing" manner (either every layer of the RoBERTa transformer is sparsified, or none are). Typically,

we have observed vastly decreased model performance when using sparsification in every layer. In Fig. 6, we apply sparsification to only *some* of the layers, starting with the **final** layer and working forward (in the validation set). Note: in the case of 0 layers, we are simply showing baseline performance for unmodified RoBERTa. Our goal is to observe if we can apply some sparsity (for improved inference performance) while still preserving good performance.

6 Discussion and Analysis

6.1 Limitations of Approach

It is clear that once a model is *entirely* sparsified with our approach, the effect on perplexity is catastrophic (RoBERTa goes from < 15 perplexity before sparsification to > 2000 after sparsification, as seen in Fig. 5, demonstrating massive inference failure). There are many potential explanations:

- Kernels are not expressive enough to represent necessary attention patterns.
- Max-based loss to replicate OR does not effectively train all kernels to be complementary.
- Neural network is not large or expressive enough to properly predict kernel mixture.
- Global thresholds are not expressive enough to capture variation in attention sparsity across layers.
- A 5-token window around the query token might not contain enough contextual information

It seems likely that expressivity concerns are at the top of the list, as we observe in training that the train loss fails to meaningfully decrease, suggesting that our model is inherently too limited to represent the true behavior of attention. We have observed that different kernel design procedures *do not* seem to drastically change end task performance, suggesting that the precise design of kernels is perhaps not as important as the selection module trained for selecting kernels.

We have observed that perplexity exponentially increases as the number of sparsified layers increases (Fig. 6). As perplexity is an exponential measure, this is not unexpected, but it suggests that each layer, individually, does not alter the model activation by too much, but when these layers are combined, the cascading effects of slightly different activation at each layer build up and destroy performance. In practice, we might be able to reduce the execution of 2 or 3 layers to sublinear complexity with our sparsification procedure, without paying too much of a perplexity penalty.

The effect of cascading error propagation is enhanced when one considers that the model's training distribution is precisely the set of intermediary embeddings produced by *unmodified execution of RoBERTa*. In other words, our model is completely unprepared to handle the adjusted distribution of intermediary embeddings that will be produced when prior layers are sparsified under the model.

6.2 Improvements to Approach

To combat the issue of an inexpressive neural network model, we might increase the size of the neural network and increase its expressivity accordingly. We cannot increase the width of each hidden layer without quadratically expanding the number of parameters and FLOPs required for evaluation and training, but we can increase the depth to linearly increase FLOPs while supporting more complex behavior. To improve training and overcome the error propagation problem, we must integrate the training of our kernel predictor module into the overall training of RoBERTa, simultaneously updating weights for both by back-propagating through the entire network. This will be computationally expensive, but will help avoid the issue of feeding an entirely different distribution on intermediary embeddings to each layer's model at training and evaluation times. It will also accustom the RoBERTa model to the exact level of sparsity our module produces. We also aim to pursue a modified loss function which operates independently for each kernel, to avoid the issue of gradients only flowing through the maximum kernel coefficient (as we currently experience).

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[3] Yao-Hung Hubert Tsai, Shaojie Bai, Paul Pu Liang, J. Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov. Multimodal transformer for unaligned multimodal language sequences. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6558–6569, Florence, Italy, July 2019. Association for Computational Linguistics.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[5] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[6] Marcos V. Treviso, António Góis, Patrick Fernandes, Erick R. Fonseca, and André F. T. Martins. Predicting attention sparsity in transformers. *CoRR*, abs/2109.12188, 2021.

[7] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020.

[8] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *CoRR*, abs/2009.06732, 2020.

[9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

[10] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *CoRR*, abs/2001.04451, 2020.

[11] Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. *CoRR*, abs/1909.00015, 2019.

[12] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016.

[13] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *CoRR*, abs/2110.12894, 2021.