Document Summarization using Transformers with Information Selection Layer

Jean Paul ISHIMWE
African Institute for Mathematical Sciences
Ghana, Accra
jishimwe@aimsammi.org

Edouard Grave
Facebook AI Research
France, Paris
egrave@fb.com

Abstract

Recently, transformers have obtained strong performance on many different tasks in natural language processing, leading to their wide adoption in the field. One of the main building blocks of transformers is the self-attention mechanism, used to learn long term dependencies. A limitation of transformers for summarization is their lack of information selection, as all representations from the encoder output are routed to the decoder for generating summaries. In this work, we propose a modified attention and gating mechanisms in the transformer that are capable to tackle these problems. The information selection layer is a gating mechanism that we introduced between encoder and decoder modules of the transformer to learn relevance scores of the words in the input document. The gate scores are then used to select important information from the document before generating summaries. Via jointly training the transformer and the selection layer, we regularized the gate scores with the penalty lambda (λ) for inducting different levels of sparsity and threshold these scores to remove tokens from the document. Our experimental results obtained using the NEWSROOM datasets show that we can drop 77% of the tokens while the performance remains the same as using the full document. Furthermore we show that our model outperformed the baseline models on NEWSROOM dataset when the model is regularised by $\lambda = 0.1$ with threshold of 0.1 and sparsity rate of 58.20%.

1 Introduction

Document summarization is a natural language generation task that deals with compressing long documents to concise and fluent summaries, while maintaining the general meaning of the original text. In summarization, there are two broad categories of techniques: extractive and abstractive methods (Kasture et al., 2014). In extractive approaches, the generated summaries are based on content extracted from the original documents, such as sentences. This extracted content might be simplified, for example by removing words or phrases. This approach may produce good summaries, however, it suffers from redundancy in the produced sentences (Hou et al., 2017; Shi et al., 2018). On the other hand, abstractive approaches are based on semantic understanding of the original text, before generating summaries from scratch. In particular, these methods first build an "abstract" representation of the original document (such as dense vectors in the case of neural networks), which is then used to generate summaries. Nowadays, abstractive summarization produces high quality, grammatically correct and sounds summaries which are far better than extractive approach (Li et al., 2018; Hou et al., 2017; Rossiello, 2016; Shi et al., 2018).

Recently, sequence-to-sequence models (Seq2Seq), based on recurrent neural networks (RNN) have received widespread attention for text generations tasks like machine translation (Bahdanau et al., 2014) and text summarization (Nallapati et al., 2016a). The Seq2Seq framework has been successful by obtaining state-of-the-art results over the former approaches (Rush et al., 2015; Nallapati et al., 2016b). Despite these good performances, Seq2Seq models/approaches still generally have drawbacks that need to be addressed. First, they are not able to capture long term dependencies for long sequences of tokens. Second, their sequential nature slows down training (Cai et al., 2019). Vaswani et al. (2017) proposed a new Seq2Seq architecture, called transformer, to address these issues. At train time, this model is able to process sequence of tokens in parallel, which makes it more efficient and flexible to tackle long term dependencies. However, it is unfortunate that the transformer and its variant forms for summarization still have some limitations: (1) The model doesn't use the most importance information in the summary because sometimes it focuses on irrelevant information instead of the most important information (2) it is slower at inference time (Zhang et al., 2020; Zhang and Liu, 2020).

In this work, we address the above issues in the transformer, which will make it more efficient for generating fluent summaries.To this end, we developed a transformer that uses a gating mechanism with the aim of removing the irrelevant information and prune them out from the encoder outputs before generating summaries. Transferring knowledge from pre-trained models has been proven to improve the performance on downstream tasks (Liu and Lapata, 2019; Beltagy et al., 2019). Thus, we fine-tuned T5 transformer by Raffel et al. (2019) with some modification; the gate layer was added to the T5-transformer encoder and the softmax in the cross-attention mechanism was replaced with a masked softmax to include the effect of the gates. The gate layer receives the last hidden state from encoder and then learn a score corresponding to each token in the document. The same learned gates are shared across all attention heads via the encoder-decoder attention which can favor the complete removal of some tokens from the document with zeros gates (i.e $g_i = 0$). For inducing gates sparsity, we added an L_1-norm regularization on the gates scores to the objective function.

2 Related works

The Sequence-to-sequence (Seq2Seq) framework was introduced for machine translation (MT) (Bahdanau et al., 2014). This framework has also been successfully applied to image captioning, conversational models (Vinyals et al., 2015; Li et al., 2016) and text summarization. The success of attention-based neural MT opened up a broader way of research in summarization tasks. Rush et al. (2015) first applied encoder-attention-decoder framework for sentence simplification. With no recurrence in their framework, they proposed a feed-Forward neural network (FFNN) with neural network language model as a decoder and attention-based encoder. Nallapati et al. (2016b); Rossiello (2016) modified the FFNN model and replaced it with a recurrent neural network (RNNs) for the encoder-decoder framework. This model was improved by incorporating a copying mechanism to tackle repetition problem from the output summaries by Gu et al. (2016).

The previously mentioned models were limited to encoding document at the sentence level and generally suffered from generating short summaries. Hou et al. (2017) improved the framework with a joint attention to deal with repetitive contents from the output summaries and can encode a single document for summarization. Although the performance of Seq2seq made abstractive document summarization task more viable, it still encodes and generates summaries that contain redundant and not concise information. To address these issues, Gehrmann et al. (2018) introduced a content selector in neural abstractive summarization network. The model selects salient information from the document, thus document compression and then generates summaries using the selected information. This content selection process is defined in a similar way to extractive summarization approach, which seems to be equivocal when a piece of text to be extracted from the original document does not have proper parts that convey the main idea of the text (Fu and Liu, 2018). Summarizing with salient information is key in summarization tasks. Li et al. (2018) instead of using extraction approach, defined an information layer in the Seq2seq framework. The building blocs of this layer are the gated global information filtering and a local sentence selection. The gate network first filter out globally unnecessary information and then the later select important sentences while generating summaries. Xu et al. (2019) improved the information extraction approach by using dynamic selective gate which filters out low-dimensional document encodings based on the context of the decoder state.

Though Seq2seq neural models marked an important improvement over traditional approaches like statistical machine translation (Banko et al., 2000) and phrase based machine translation (Wubben et al., 2012), they are slow to train and inherently incapable to capture long term dependencies between a token and its counterpart tokens which results with a lack off a good encoded representations (You et al., 2020; Cai et al., 2019). The recent Seq2seq Transformer (Vaswani et al., 2017) demonstrated

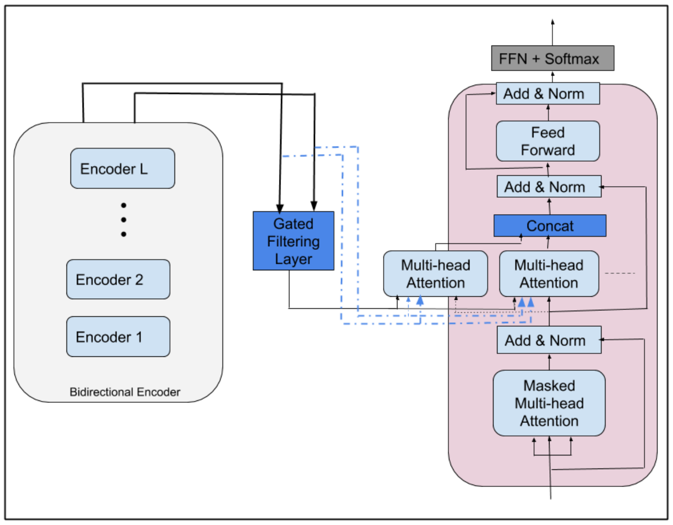

Figure 1: The model overview with the gate filter layer in the middle of the encoder and decoder modules

to be able to address those issues. You et al. (2019) incorporated in a transformer a focus-attention mechanism to help document comprehension when encoding and a saliency selection network that applies gates for extracting salient information from the document's encoding. Cai et al. (2019) used a gated linear unit to filter out information from sequential context and local importance features. Zhang and Liu (2020) proposed a selective gate that get core information from a long sentence representation together with its n-gram features. In terms of sparsifying the attention weights, Zhao et al. (2019) proposed a sparse attention mechanism in the Transformer that can select top-k contributing components in the attention score and the rest are discarded. Unfortunately all these approaches access to all encoder outputs during training and evaluation. Like in Zhang et al. (2020) our approach is to remove unnecessary information before generating the summaries.

Zhang et al. (2020) improved over Zhang and Liu (2020); Zhao et al. (2019) by zeroing out the tokens in the encoder outputs before generating summaries at inference time. They achieved this by learning random variable gates from a HardConrete distribution with an L_0Drop layer. This layer was used for pruning off the irrelevant encoder outputs. Their approach uses different gates across all the multi-heads attention. The key difference from their approach to our method; we built on their approach and learn directly the same gates for all the attention heads from encoder outputs which can allow us to remove elements from the encoder outputs. On the other hand, their approach has different sparsity pattern for all the attention heads, which can not afford of removing any elements from the encoder outputs.

3 Model

In this section we are going to give details about our model. As shown on Fig.1, our model is similar to the original Transformer architecture with both encoder and decoder components. The main difference is the gated filter layer incorporated between the encoder and decoder modules.

3.1 Transformer background

The Transformer model proposed in (Vaswani et al., 2017) relies on the self-attention mechanism to capture sequence information. The adoption of scaled dot product attention enables it to learn the dependencies between tokens within the sequence and provide a good semantic representation of the sequence. This mechanism uses both the encoded representations of encoder hidden states and decoder hidden states or output. The M key and value pairs are obtained from the encoder output, where M is the input sequence length, whereas queries come from the decoder hidden states.

The attention mechanism is defined as follows:

$$\text{Attention(Q,K,V)} = \text{AV} = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where Q,K,V are queries of dimension d_k, keys of dimension d_k and values of dimension d_v respectively.

To make this model more powerful, it uses the multi-head attention mechanism which enables it to capture different patterns. Using linearly projected queries, keys and values it can compute N-attentions or heads independently and then concatenate their output. This mechanism uses linear combination of vectors learned independently which is more powerful to learn a pattern from the sequence of words than using a single self-attention which provides the output results based on using a linear combination of all input sequences. In the context of multi head attention, queries, keys and values are all computed from the input sequence representation.

Given a list of queries, keys and values, we can compute the multi-head attention as follows:

$$\begin{aligned} \text{MultiHead}\left(Q, K, V\right) &= \left[\text{head}_1, \text{head}_2, \cdots, \text{head}_N\right] \text{W}^0 \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \textit{ and } [\,] = \text{concatenation} \end{aligned} \tag{1}$$

The W^Q, W^K, W^V, and W^0 are learnable parameter weights matrices with d_k, d_k, d_v, and model dimension (d_{model}) dimensions respectively. Seq2seq Transformer contains the multi-head attention mechanism in both encoder and decoder parts which makes it more efficient than Seq2seq RNNs in computation.

3.2 Information Selection

Selecting information is required for document summarization. Doing this highlight the most essential sentences, or words, from the document to generate a good summary. To achieve this, we want to remove part of the elements in the encoder outputs, relying on a soft gating mechanism that selects a subsets of the inputs tokens. To do this, three components are required: Encoder, Gates and Decoder. In this part, we describe these three components.

3.2.1 Transformer: Encoder

Learning document representations by Transformer's encoder proved to perform better than the traditional encoding paradigms (Devlin et al., 2018). The encoder consists of K identical layers and each layer has two main sub-layer components; the first is the self-attention mechanism sub-layer and the second is the feed-forward sub-layer. Each of this sub-layer is rounded with a residual connection and the resulting output passes through the layer normalization.

Given an input sequence $\mathbf{X} = (x_1, x_2, \cdots, x_M)$ where $x_i \in \{1, \cdots, V\}, i = 0, 1, 2 \cdots, M$ is the sequence of token indices with the sequence length M and V is the vocabulary size, the encoder learns the sequence representation into a sequence of vectors $\mathbf{H} = (h_1, h_2, \cdots, h_M)$ where $h_i \in \mathbb{R}^{d_{model}}, i = 0, 1, 2 \cdots, M$ and use it to compute the gate scores as described in subsect.3.2.2.

We can represent the encoder stack of a transformer as follows:

$$\mathbf{H} = \texttt{ENCODER}(\mathbf{X}) \tag{2}$$

where $\mathbf{H}$ is the encoder output sequence representation of input sequence $\mathbf{X}$.

3.2.2 Gate layer

It is difficult for transformers to distinguish which are salient information from unnecessary information in a long document. This results with irrelevant information in the abstractive document summarization (Zhang and Liu, 2020). Due to this effect, removing irrelevant information from the document plays an important role in abstractive document summarization. To tackle this issue in the transformer, we introduced a gating mechanism that receives the document representation $\mathbf{H}$ from the encoder and provides a relevance score for each token in the whole document. This reduces the effect of irrelevant information and imposes to have a short or compressed document to summarize from.

In general, for a given encoder output sequence $\mathbf{H}$, the gate scores G $= (g_1, g_2, \cdots, g_M)$ for this sequence are computed as follows:

$$G = \sigma\left(\mathbf{H}w^T\right), \tag{3}$$

where $w \in \mathbb{R}^{d_{model}}$ and σ are the weight matrix and sigmoid activation function respectively. Each of the $h_i \in \mathbb{R}^{d_{model}}$ has its corresponding score g_i.

The generated scores from sigmoid are in $[0 - 1]$ where values assigned with scores close to zero are considered to be irrelevant information that can be dropped off from the document and retain only tokens with high scores. To force some of the values of the gate scores to be equal to zeros, we regularized the gate scores.

After computing these scores, they can be used in the attention module of the decoder part for removing irrelevant tokens before generating summaries.

3.2.3 Transformer: Decoder

The decoder is used to generate the summary based on the encoded input sequences. Like in encoder, the decoder consists of K identical layers wherewith each layer has an additional sub-layer. The three sub-layers belonging to each layer are; the masked self-attention mechanism, multi-head cross-attention and the feed-forward sub-layers. The multi-head cross-attention mechanism receives the encoder outputs as well as the target representations. To prevent the transformer from attending to the future words during training a mask is applied in the attention mechanism. Each of this sub-layer is rounded with a residual connection and then followed by the normalization layer.

We can represent the modified decoder as follows:

$$\mathbf{Y}_l = \texttt{DECODER}(\mathbf{H}, \mathbf{Y_0}, G), \tag{4}$$

where $G \in \mathbb{R}^{\text{Bx 1 xMx1}}$, $\mathbf{H} \in \mathbb{R}^{\text{BxMxd}_{model}}$, $\mathbf{Y_0} \in \mathbb{R}^{\text{BxMxd}_{model}}$ represent the gates,encoder last hidden states and shifted target sequences respectively. The dimensions' symbols stand for B: the batch-size, and M: sequence length.

The regular attention mechanism in transformers is computed as follows;

$$\begin{aligned} \alpha_i &= e^{qK_i^T} / \sum_{j=1} e^{qK_j^T} \\ \text{c} &= \sum_{i=1} \alpha_i V_i, \end{aligned} \tag{5}$$

where c is the weighted sum of the values for the i^{th} query and α_i is the weight for the i^{th} token.

The attention mechanism computes the weighted sum attention over all value tokens. Our main goal is to remove some of the elements in the sum of eq.5 which are irrelevant when generating summaries. To do so, we want to add a gating in the eq.5 which gives,

$$\begin{aligned} \alpha_i &= e^{q\text{g}_i K_i^T} / \sum_{j=1} e^{q\text{g}_j K_j^T} \\ \text{c} &= \sum_{i=1} \alpha_i \text{g}_i \text{V}_i, \end{aligned} \tag{6}$$

where all the α_i distribution weights $(i.e\ \ i = 1, 2 \cdots M)$ sum to one.

The computed last hidden state from the decoder stack $\mathbf{Y_l}$ is finally passed through a feed-forward layer followed by a softmax layer to compute probability distribution of the next token.

To make the decoding process faster, we used a modified multi-head attention mechanism that computes attention weights distribution for queries using only relevant keys. During inference and test time we threshold all the gates values below a threshold value to zero. This will mask out all the keys and values pair that align with gate scores values that are exactly zero.

To reduce the irrelevant information from the documents before generating concise summaries, we revised the decoding process as shown in Algorithm.1, that gives a compressed sequences to summarize from. We used the notation $\odot$ to denote the element-wise multiplication, $\mathbb{1}$ refers to a tensor of ones and $\hat{.}$ over any variable represented here by . denotes the extracted values from their original values. We first start by explicit thresholding the gate scores for creating a mask. Everything below the threshold is assigned to 0 otherwise 1. Then, we apply the gate mask on the gate scores to close the gate scores of the corresponding tokens. The use of gate scores on the keys and values, it completely zeroes out their corresponding tokens.

For each token with a closed gate (ie. $g_i = 0$) is worthless for summary generation, it is however removed from the candidates to use while generating summary. we finally apply the attention weight learned by the softmax on the values to get how much are they important on the query provided. The application of the same gates across all different attention heads (implies using the same learned pattern) enable the removal of some elements in the document which results with shortened document encodings, that reduce the runtime of vanilla transformer decoder, that scales linearly with the document size. As with human generated summaries, not all the information is used from the document to get a summary, so can filtering out irrelevant information from the document in transformers provide promising concise summaries.

Algorithm 1: Decoding algorithm during inference and test time for encoder-decoder multi-head attention mechanism

```
Input:
  · Queries tokens $(Y_0) \in \mathbb{R}^{B \times S \times d_q}$
  · Encoder output(H) $\in \mathbb{R}^{B \times S \times d_{model}}$
  · Gates scores (G) $\in \mathbb{R}^{B \times 1 \times S \times 1}$
Output: Context for the queries

ATTENTION($g,H,Y_0$):
    a) Thresholding gate scores over threshold (t)
    • $T \leftarrow G > t$                                          ▷ T is a boolean tensor
    • $G_{mask} \leftarrow T \odot \mathbb{1} \odot$ attention mask
    • $\hat{G} \leftarrow G_{mask} \odot G$
    b) Attention with gate scores
    • $Q,K,V \leftarrow Y_0W_q, HW_k, HW_v$                         ▷ projected queries, keys and values
    • $\hat{K}, \hat{V} \leftarrow K \odot \hat{G}, V \odot \hat{G}$    ▷ Extracted keys and values
    • $Score \leftarrow Q\hat{K}^T$
      Softmax for $i^{th}$ query
    • $\alpha_i \leftarrow \frac{e^{Score_i}}{\sum_{j=1} e^{Score_j}}$
    • $c_i \leftarrow \sum_{n=1}^{N} \alpha_n^i \hat{V}_n$          ▷ where $c_i$ is the context vector for $i^{th}$ query
    return $c_i$
```

3.3 Objective Function

For the target sequence $\mathbf{Y_0} = (y_1, y_2, \cdots, y_L)$ and input document sequence $\mathbf{X} = (x_1, x_2, \cdots, x_M)$ where L and M are the numbers of tokens of the document and summary, the transformer encodes the sequence $\mathbf{X}$ and use it to predict the most likely next token using the maximum likelihood estimation approach. To have an effective performing system, we give a transformer to learn from the document by minimizing the negative log-likelihood loss between predicted sequence and the sequence $\mathbf{Y_0}$ pair during training. The encoder output (i.e last hidden states) is first used to compute the gates scores, then both hidden states and the scores are fed through the encoder-decoder-multi-head attention mechanism to the decoder. By jointly training vanilla Transformer encoder (T5 Transformer) and the gates layer, the gates are implicitly pushed to zero via the sigmoid function.

The loss and the problem of predicting next token is mathematically formalized as follows:

$$\begin{gathered}
\mathrm{p}\left(Y_0 | X; \theta\right) = \prod_{t=1}^{L} p\left(y_t | y_{<t}; X; \theta\right) \\
\mathrm{L}_{MLE} = -\frac{1}{|\mathcal{D}|} \sum_{Y_0, X \in \mathcal{D}} \log \mathrm{p}\left(Y_0 | X; \theta\right);
\end{gathered} \tag{7}$$

where $\mathcal{D}$, $|\mathcal{D}|$ and θ are dataset, dataset size and set of model parameters respectively.

To force this model to learn good scores distribution and induce some sparsification effect on the gate scores, we changed the minimization loss in eq.7 and add the L_1-norm regularization term on the gate scores defined as follows:

$$\mathcal{L}_{gates} = \sum_{i} |g_i| \tag{8}$$

Thus far the general objective to train our transformer is by minimizing both the gates loss and the negative log-likelihood loss. The overall training loss is:

$$\mathrm{L} = \mathrm{L}_{MLE} + \lambda \mathcal{L}_{gates}; \text{ where } \lambda \text{ is a hyperparameter.} \tag{9}$$

4 Experiments

In this section we give a detailed dataset description, model settings, evaluation metrics used and the results obtained in this project.

4.1 Datasets

We performed our experiments using the newsrooms summarization dataset (Grusky et al., 2018) for both training and validation. The Newsrooms dataset contains 1.3 millions article-summary pairs. It is high quality data collected from using social media and search engine metadata with human written summaries. It is originally splitted into train, development and test sets $(82\%, 9\%, 9\%)$. We processed both train, development and test sets, and every document with sequence length less than 100 or summary sequence length less than 10 was dropped from the data-frame. The input document was truncated to 700 tokens.

4.2 Model settings and baselines

We initialized the model with T5 transformer for conditional generation (T5-base) pre-trained weights. We used a standard transformer architecture which has both the encoder and decoder blocks with the gate layer in between. We experimented with the base settings of the T5-base transformer of Raffel et al. (2019); the encoder and decoder each has 12 blocks, the dropout probability is set to $p = 0.1$, the feed-forward layer in each block has an output dimension of $d_{ff} = 3072$, the keys of $d_{kv} = 64$ dimension, and with all the attention mechanisms of 12 heads. The gate layer was initialized with the default initialization in the linear layer. The hyper-parameters λ and the threshold were obtained by doing grid-search over a set of range of values using the weights and biases (wandb) sweeps developer tools for machine learning (Biewald, 2020).

We evaluated our model against the following baselines using different summarization strategies:

1. **Pointer-N**: Grusky et al. (2018) trained pointer generator model on NEWSROOM dataset which uses a copying mechanism to copy a token and generate a token as well as keeping track of what has been extracted so far.
2. **Summary Loop 24** This model uses unsupervised abstractive summarization approach based on maximizing both coverage and fluency for a given length. This technique encourages the use of important key terms masked from original document and are filled in by a coverage model from using the generated summary (Laban et al., 2020).
3. **Modified P-G** Shi et al. (2019) re-implemented pointer-generator model by adding some minor changes to the neural network to fit in the neural abstractive text summarization (NATS) toolkit.
4. **Pointer-Transformer Language Model (P-TLM)** Summarization task is done into two steps: extractive step and an abstractive step. First the model extracts important sentences from the paper and then the transformer language model is conditioned on relevant information before generating summary (Subramanian et al., 2019).
5. **Seq2Seq + Attention** Model proposed by (Rush et al., 2015) that uses feed-forward neural networks for both encoding and decoding with a local attention mechanism between them.

4.3 Evaluation

The evaluation metric ROUGE (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004) was used to evaluate the model performance. It automatically measures the quality of summaries by computing the overlap between the summaries generated by the model and the reference summaries written by humans. The performance is measured based on N-gram overlap and longest common subsequence (LCS) statistics. In all the experiments reported here, we used the set of ROUGE metrics; ROUGE-1, ROUGE-2 and ROUGE-L evaluated on a subset of size 1k during inference for both validation and test sets. Moreover, we also used the perplexity metric which is the inverse probability on the validation set during training and used it to save checkpoints for the best model.

4.4 Results and Discussions

We run experiments using the NEWSROOM datasets. We first fine-tuned T5-base transformer with the gate layer added on top of the encoder's stack. As shown from eq.9 for sparsifying the gates, we trained the model by exploring different values of λ ranges $[0.0, 0.7]$ with a step size of 0.1. We also introduced a new hyper-parameter *threshold* during inference that explicitly thresholds the gates less than that particular value. Thresholding in this manner is equivalent to removing tokens from the transformer decoder input representations before generating the summaries. The threshold is set in the range $[0.0, 1.0)$ and with a step size of 0.01. To determine how much we have sparsified, we calculated the sparsity ratio as the ratio of the total number of tokens dropped (when *gates = 0*) and the total number of tokens in the source encodings. As illustrated from Figure.2 we trained different models with only the difference is the penalty incurred on the gate scores to explore disparate levels of sparsity. To our best knowledge the results shows that the regularization is helpful even though a none regularized model can still drop tokens from the source words. To the rest of the next pages unless said the results were computed using the best model regularized using $\lambda = 0.1$.

Table 1 shows the systems performance results on both development and test sets. Training a transformer on this benchmark substantially increases the performance for all *F1*-Rouges scores over all the baseline models on test set and on development set outperforms neural based model Pointer Network trained on this dataset. The use of gating layer in the transformer potentially helped to drop tokens and the approximate rouge scores can be achieved. With $\lambda = 0.1$ the sparsity rates are 58.20% and 57.61% for test and development datasets respectively. We report R-2 as our primary metric. Our transformer outperformed the modified P-G by 0.58 ROUGE-2 points, transformer language model with pointer by 8.95 ROUGE-2 points, and the Pointer-N by 15.71 ROUGE-2 points on test set and 11.05 ROUGE-2 points on development set as shown in Tables.1a,1b respectively.

Table 1: The performance (Rouge-1, Rouge-2, Rouge-L) of the systems on the NEWSROOM datasets. All the models marked with * evaluated on full NEWSROOM dataset whereas our transformer evaluated on a subset of it with $\lambda = 0.1$ and threshold $= 0.1$.

(a) Development set released (T)

NEWSROOM - T				
Model	R-1	R-2	R-L	R-2 (no thresholding)
Pointer-N*	26.27	13.55	22.72	-
Our transformer	**37.73**	**24.61**	**32.94**	**24.34**

(b) Test set released (T)

NEWSROOM - T				
Model	R-1	R-2	R-L	R-2 (no thresholding)
Pointer-N*	26.02	13.25	22.43	-
Summary Loop 24*	27.0	9.6	26.4	-
Modified P-G*	39.91	28.38	36.87	-
P-TLM*	33.24	20.01	29.21	-
Seq2Seq + Attention*	5.99	0.37	5.41	-
Our transformer	**41.11**	**28.96**	**36.88**	**28.26**

Table 2: The performance results on different types of pruning out encoder outputs using gates with $\lambda = 0.1$ and $\lambda = 0.2$ for same and different gating mechanisms on the subsets of NEWSROOM test set released (T) respectively. Sparsity is the dropped tokens rate; where the threshold equals 0.00% implies the summary was generated with all encoder output tokens.

Model	**Thresholds**	**Sparsity**	**R-1**	**R-2**	**R-L**
T5-transformer using same gates	0.0	0.00%	40.13	28.26	36.09
	0.04	33.18%	40.11	28.17	35.97
	0.08	49.15%	40.02	28.10	35.85
	0.1	58.20%	41.11	28.96	36.88
	0.2	88.51%	36.65	23.40	32.77
T5-transformer using diff gates	0.0	0.00%	41.26	29.26	37.13
	0.04	39.15%	40.47	28.47	36.42
	0.08	52.16%	40.29	28.26	36.17
	0.1	57.20%	40.50	28.40	36.41
	0.2	75.76%	40.34	28.30	36.30

The transformer gives better results. Using a transformer does improve the performance on downstream tasks due to its capability of handling long sequence compared to RNNs. As shown from Table.1 the transformer model outperforms the Seq2seq neural network across all the F1-measure scores. The explanation for this result, comparing the difference in the scores to the baseline, is twofold: (1) using a transformer instead of RNNs (2) using a model that was pre-trained on large amount of unlabeled data. While capturing the interdependence between tokens in a long sequence provides a good signal to the model, the Seq2seq RNN based model on the other hand, has one limitation what is referred to as information bottleneck and due to this long sequence becomes problematic and also the last encoded words are given high importance which can results to poor performance.

Table 3: The effect of regularizing the gates scores.

NEWSROOM Devset			
Penalty (λ)	Threshold	Sparsity rate	R-2
0.0	0.1	41.74%	24.05
0.1	0.1	57.61%	24.61
0.2	0.1	72.23%	23.35
0.4	0.1	78.04%	21.70
0.7	0.1	80.52%	19.34

Results from models trained with different encoder outputs pruning objectives are shown in Table.2. *What can be the effect of pruning out encoder outputs using same or different gates across multi-attention heads?*. To find the effect of gates, we experimented with two approaches: (1) applying the same gates on all attention heads (2) learn a separate gate on each of the attention heads. We can see that the model trained using the same gates scores on all attention heads drops drastically as long as the thresholds increases. This is because the model has sufficiently learned the same pattern and can be easily to remove tokens from the document. Using different gates on the other hand, it is hard to drop tokens since it has learned different patterns across all the heads. The generated scores are stable compared to the transformer using same gate scores for all heads. This proves our hypothesis of removing irrelevant information from the document.

How better is the encoder stack for learning the gate scores? We visualized the distributions of the gate scores computed from the last hidden states of the encoder using a transformer trained with a regularization on the gate scores of $\lambda = 0.1$. From Figure.3 shows the gate scores distributions where each word from the document is assigned with a relevance score. The gating mechanism of the transformer assigns high scores on meaningful words and numbers in the document like "bank" and "6.4", and lower scores on irrelevant words which enables the transformer to route relevant information from the source document. We obtained this result because we fine tuned a large scale pre-trained transformer with additional jointly trained gate layer that learns the scores from good representations of the transformer encoder.

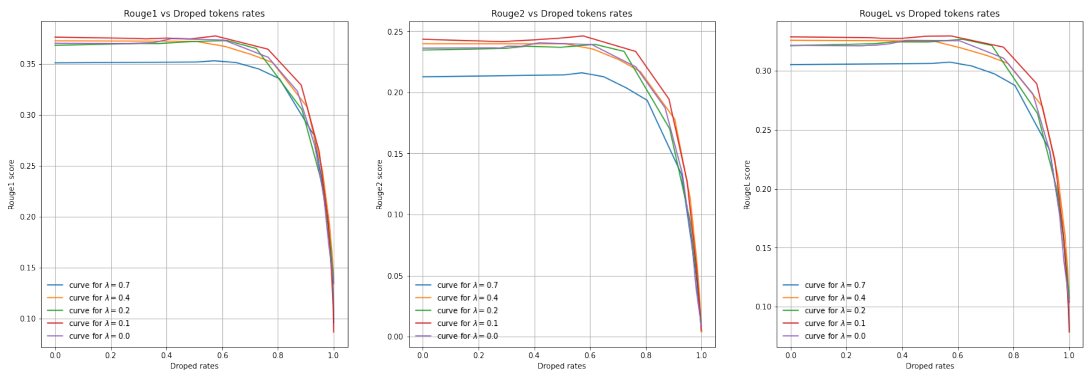

Figure 2: Comparison of models with different regularization parameter (λ).

As illustrated from Figure.4, dropping tokens before generating summaries does not largely degrade the model performance, which indicates the usefulness of compressing information by pruning out encoder output before generating the summaries. We can remove 77% of the tokens and the generation performance only drops by -2.28 ROUGE-2 score. This can prove that not all the tokens in the document are really important while summaries are getting generating. Removing irrelevant information can therefore lead to salient and fluent summaries.

The impact of regularization on the gate scores. We first trained models using different regularization values to see how it affects the performance of the model. As shown from Table.3, regularization definitely has an impact on the values of the gates which concurrently affects the performance of the model. Interestingly, without regularizing the gates scores (i.e regularization penalty = 0.0) we

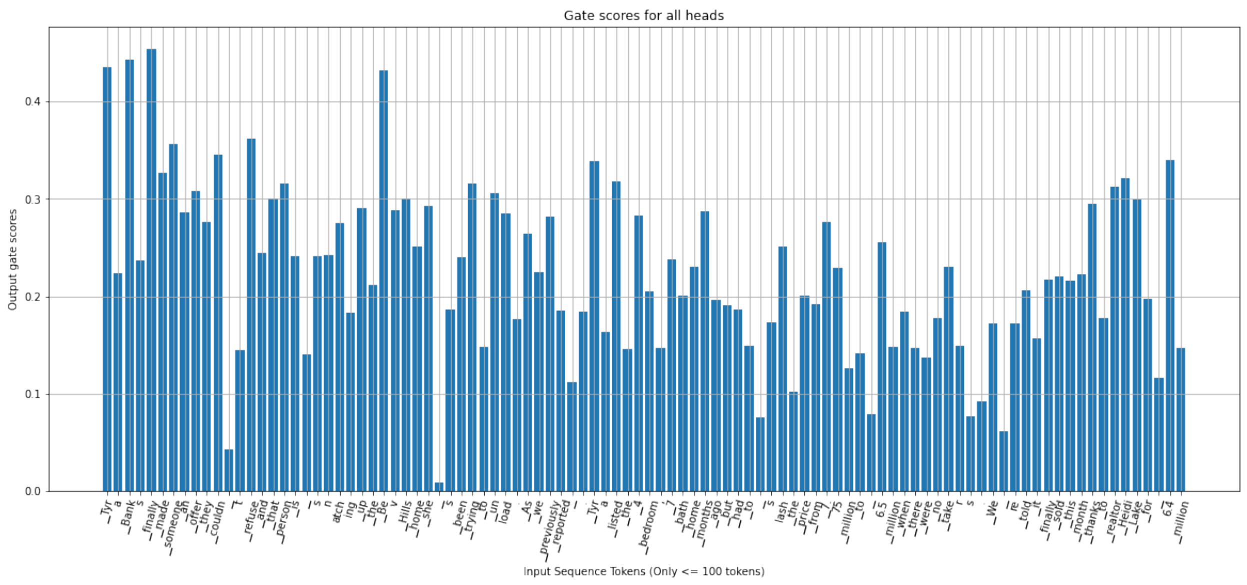

Figure 3: Illustration of gate scores computed using NEWSROOM test-set. The blue bars indicates the tokens' scores that are assigned by the gating mechanism according to their relevance in the document. The gating mechanism of the transformer assigns high scores on meaningful words and numbers in the document like "bank" and "6.4", and lower scores on irrelevant words.

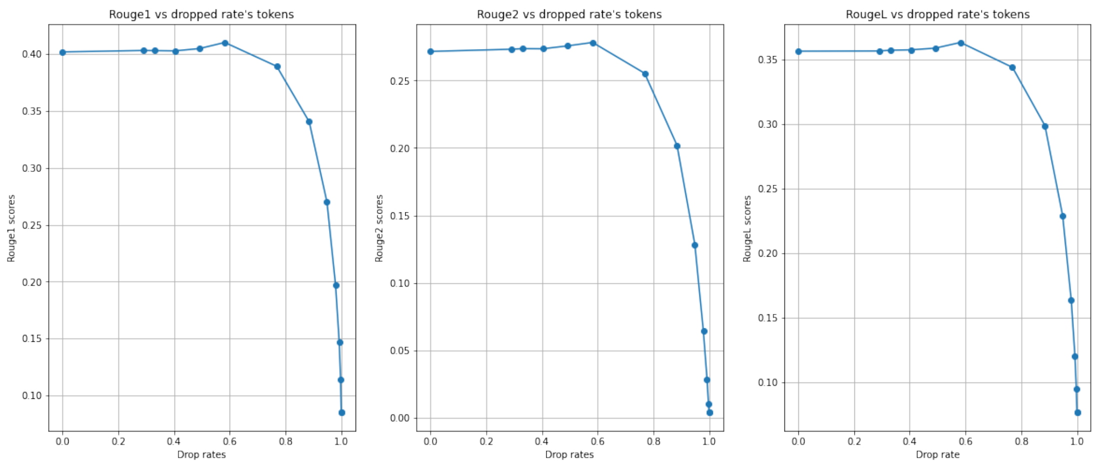

Figure 4: Generated scores on y-axis as function of sparsity rates on x-axis evaluated on NEWSROOM test-set. By pruning out 77% of the source encoding tokens doesn't hurt the general performance of the model

can still learn them pretty well. When we don't regularize and want to remove all the tokens values less than or equal to the threshold equals 0.10, 41.74% of the tokens are removed which implies that 41.74% of the gate scores are smaller than 0.1 with no need to force the model to push them towards zero. This is because the model has already learn to push the scores towards zeros to filter out words from encoder representations whereby we anticipated the gate scores to be close to one when no regularization is used to keep all the tokens and still be able to perform regular attention mechanism on all the tokens. This shows that regularization was not important. However, we still want to drop as much as irrelevant information we regularized. When we regularized the model to fairly push down the gate scores using the penalty of 0.2 we were able to achieve comparable performance to none regularized gate scores, that is a change of -0.69 R-2 score and 72.23% of the tokens were removed from the documents' representations. To prove that regularization is also helping we illustrated it from Figure.2, we can see that smaller values of regularization (i.e like $\lambda = 0.1$) have good performance at a certain degree of dropped tokens across all ROUGE scores than the others. The largest value of regularization (i.e like $\lambda = 0.7$) has a drastic drop on the ROUGE scores which indicates that it affects the model performance and we shouldn't too much regularize it.

5 Conclusion

In this report, we saw that using gating mechanism in the transformer enables to prune out redundant information from the document before generating summaries. By reducing redundancy from encoder output proves that it can be shortened and summarize from salient information, which significantly improves the performance of the document summarization task. This was achieved by jointly optimizing T5-transformer and the gate layer which resulted with a better performance than our baseline models. The gate layer learns similar pattern for all the heads which enables to explicitly drop tokens from the document. From a series of experimental results we show that 77% of the source tokens can be dropped off and result with a comparable performance. Aside from improving the performance and the quality of the generated summaries, the transformer with our gating mechanism can boost the decoding speed up of the transformer once a number tokens from the encoder output are pruned out. This proves our assumption that removing irrelevant information from the document plays an important role in abstractive document summarization.

Acknowledgments and Disclosure of Funding

I would like to express my gratitude to my supervisor, Edouard Grave for his guidance, advice and support throughout this work. I also wish to extend my special thanks to the management team of African Masters in Machine Intelligence (AMMI) for their unceasing encouragement over the whole academic year.

References

NR Kasture, Neha Yargal, Neha Nityanand Singh, Neha Kulkarni, and Vijay Mathur. A survey on methods of abstractive text summarization. 2014.

Liwei Hou, Po Hu, and Chao Bei. Abstractive document summarization via neural model with joint attention. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 329–338. Springer, 2017.

Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. Neural abstractive text summarization with sequence-to-sequence models. *arXiv preprint arXiv:1812.02303*, 2018.

Wei Li, Xinyan Xiao, Yajuan Lyu, and Yuanzhuo Wang. Improving neural abstractive document summarization with explicit information selection modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1787–1796, 2018.

Gaetano Rossiello. Neural abstractive text summarization. In *DC@ AI* IA*, pages 70–75, 2016.

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

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016a. URL http://arxiv.org/abs/1602.06023.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. 2016b.

Tian Cai, Mengjun Shen, Huailiang Peng, Lei Jiang, and Qiong Dai. Improving transformer with sequential context representations for abstractive text summarization. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 512–524. Springer, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Biao Zhang, Ivan Titov, and Rico Sennrich. On sparsifying encoder outputs in sequence-to-sequence models. *arXiv preprint arXiv:2004.11854*, 2020.

Xuwen Zhang and Gongshen Liu. Selective and coverage multi-head attention for abstractive summarization. In *Journal of Physics: Conference Series*, volume 1453, page 012004, 2020.

Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*, 2016.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*, 2018.

Junjie Fu and Gongshen Liu. A joint selective mechanism for abstractive sentence summarization. In *Asian Conference on Machine Learning*, pages 756–769, 2018.

Haiyang Xu, Yahao He, Kun Han, Junwen Chen, and Xiangang Li. Learning syntactic and dynamic selective encoding for document summarization. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

Michele Banko, Vibhu O. Mittal, and Michael J. Witbrock. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 318–325, Hong Kong, October 2000. Association for Computational Linguistics. doi: 10.3115/1075218.1075259. URL https://www.aclweb.org/anthology/P00-1041.

Sander Wubben, Antal van den Bosch, and Emiel Krahmer. Sentence simplification by monolingual machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1015–1024, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P12-1107.

Fucheng You, Shuai Zhao, and Jingjing Chen. A topic information fusion and semantic relevance for text summarization. *IEEE Access*, 8:178946–178953, 2020.

Yongjian You, Weijia Jia, Tianyi Liu, and Wenmian Yang. Improving abstractive document summarization with salient information modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2132–2141, 2019.

Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and Xu Sun. Explicit sparse transformer: Concentrated attention through explicit selection. *arXiv preprint arXiv:1912.11637*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. *arXiv preprint arXiv:1804.11283*, 2018.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Philippe Laban, Andrew Hsi, John Canny, and Marti A. Hearst. The summary loop: Learning to write abstractive summaries without examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5135–5150, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.460. URL https://www.aclweb.org/anthology/2020.acl-main.460.

Tian Shi, Ping Wang, and Chandan K. Reddy. LeafNATS: An open-source toolkit and live demo system for neural abstractive text summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 66–71, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4012. URL https://www.aclweb.org/anthology/N19-4012.

Sandeep Subramanian, Raymond Li, Jonathan Pilault, and Christopher J. Pal. On extractive and abstractive neural document summarization with transformer language models. *CoRR*, abs/1909.03186, 2019. URL http://arxiv.org/abs/1909.03186.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W04-1013.