ABSTRACTIVE TEXT SUMMARIZATION

HENRY (HANXIANG) PAN [HENRYPAN@SEAS], XINYU MA [XINYUMA@SEAS],

ABSTRACT. In this project, we study the area of abstractive text summarization in the field of Natural Language Processing (NLP). To understand what are the most important ingredients for text summarization, we plan to implement some common architectures, analyze common pitfalls, seek improvements and compare their performances. Please refer to here for a quick overview video of our project.

1. INTRODUCTION

Text summarization is the process of shorting a text computationally by creating a summary that best retains the relevant original text's information. Text summarization has applications in many domains: reading summaries instead of full texts can enhance the efficiency of information acquisition; summaries can be used to classify and index documents; text summarization can assist in other NLP tasks such as question answering and sentiment analysis. With the explosion of information nowadays, accurately and succinctly summarizing text is increasingly important.

1.1. **Contributions.** We implement several deep learning methods for text summarization. Our models have similar performances as state of the art architectures, while ours is easier to implement and is less computationally intensive. In this paper, we first introduce the background and state of the art of abstractive summarization. Next, we introduce a multi-head self-attention transformer architecture. We apply our architecture to CNN News Daily articles. Our model shows promising results and reasonable performances.

2. BACKGROUND

Text summarization is an active area of research in natural language processing. Some early works in this area are extractive [6], meaning that some scoring rules are used to select sentences. These methods are less flexible, as the gist of text may be present in multiple sentences. Nor does it reflect how human beings process information. Recent developments in deep learning revive generative methods, providing foundations for abstractive summarization. In particular, sequence to sequence language models have proved to be extremely effective for this task. Lately, sophisticated sequence to sequence architectures are beating the state of the art: Some researchers [7] employs a sequence-to-sequence oriented encoder-decoder model equipped with a deep recurrent generative decoder (DRGN), which achieves surprisingly high performances. Others [9] uses a decoder-only architecture that can attend to sequences that are much longer than typical encoder-decoder architectures used in sequence transduction. Most significantly, researchers [12] proposes a pointer-generator networks architecture, which augments standard sequence-to-sequence model with a pointer-generator network that points words from source tests to the network and applies a coverage mechanism to keep track of what has been summarized to reduce repetition. Our work initially investigates the capability of sequence-to-sequence models, specifically, we are interested in learning where the traditional recurrent architecture with a simple attention mechanism falls short. Then we would like to replace the recurrent structure with a multi-head (self) attention mechanism. This allows the model to distinguish relevant tokens from irrelevant tokens. It also overcomes the limitations that recurrent networks fail to capture long-term dependencies, since now the model will compute its attention weights from the entire input sequence as a whole, as opposed to just process token-by-token. In addition, there are other enhancements to the improved model such as positional embedding and sequence masking that are necessary for the self-attention mechanism to work properly.

3. DATASET

3.1. **CNN Daily Mail.** We used the article body, from CNN Daily Mail, as our input text and the ground truth summaries from the same source as our labels, both in plain text format. In total, there are 287227 number of articles and corresponding summaries split into 70% training, 30% validation. A separate test data contains 11490 articles and corresponding summaries.

3.2. **Guardian and Inshorts.** The article body is collected from Guardian and the corresponding ground-truth summaries are gathered from Inshorts. This dataset contains 98401 articles ("text" column) and summaries ("headlines" column) in csv format. We initially used this dataset to train our baseline model, but noticed that it lacked some variation in improving the model's generalizability, which motivated us to switch to the first larger dataset (CNN Daily Mail).

4. Approach/Methods

4.1. **Preprocessing.** Our preprocessing for the two models are slightly different, but the common preprocessing involved filtering out non-alphanumeric symbols since they have less value in training and less weight in our word embeddings, if any. Since we are working with English text, we also replaced word contractions (i.e. $don't$) with its original tokens. The model specific preprocessing steps are discussed in the respective sections.

4.2. **Features.** To tackle text-related problems, we first converted the preprocessed story and summary texts into independent tokens by splitting on white-space. We also leveraged a pre-trained word vector model, Global Vectors for Word Representation (GloVe) [10], as a way to represent text data in the form of numerical vectors, specifically we used the Wikipedia and Gigaword5 6 billion tokens 300-dimension pre-trained word vector model. By leveraging this word vector model, we created a word embedding feature for the vocabulary of our dataset. Each vocabulary is represented by a 300-dimension word vector, thus the shape of our word embedding is (vocabulary size $\times$ 300). Since the word embeddings contain linear substructures of the vocabulary space, we used it as the initial weights for the embedding layer in both our baseline model and improved model.

4.3. **Baseline Model.** Our baseline model consists of a gated recurrent auto-encoder with bidirectional attention mechanism, the architecture diagram can be found in the appendix Figure 1.

Preprocessing Since articles have varying lengths and varying summaries associated, we set a maximum token count threshold to filter out all the articles that have more than 300 tokens after going through the general preprocessing stage. Similarly for summaries, we filtered out all summaries with more than 40 tokens after the general preprocessing stage.

The process is as follows:

- The input article body x is passed to the encoder, as a whole, that will produce encoded features $enc(x)$
- The ground truth summaries y are passed to the decoder, with a initial special token prepended, in a *token-by-token* fashion
- The decoder will output the generated summary, *token-by-token*

In detail, there are two main components in this process:

Encoder: The input data is tokenized article body text x and converted into the corresponding word embeddings $emb(x)$ fed into the encoder model. The encoder consists of a bidirectional gated recurrent unit (GRU) that will selectively extract text sequence dependencies across the input text sequence in both forward and backward fashion. With a GRU, the number of parameters are greatly reduced as compared to long short-term memory networks, and the danger of vanishing gradients is mitigated as compared to general recurrent units. The encoder will output the encoded features $enc(x)$ as well as the encoder's final hidden state z_{enc}.

Decoder:

- The decoder takes in 3 inputs.
  (1) Ground truth summaries preprocessed into tokens y, where the first token is always the special reserved starting token "starttoken". This token is passed through an initial layer of the pre-trained word embedding to obtain our word feature for that particular token $emb(y)$
  (2) The encoder's final hidden state z_{enc} will be used as the initial hidden state for the decoder
  (3) The encoded features $enc(x)$ is used by the attention mechanism in the decoder. This attention concept is represented as a one-layer feed-forward network, which computes "attention scores" from the encoded features $enc(x)$ and input sequence y. This allows the decoder to only focus on a particular range in the input sequence y. These scores are then multiplied with $enc(x)$ to form the weighted sum context vector. Now the decoder will combine the context vector and gated recurrent output to pass through a Tanh nonlinearity function with a final pass through the softmax function to generate a probability distribution
- Since the decoder takes in tokens one at a time, the output is also one token at a time. Each output is a probability distribution across the entire vocabulary space, in shape $(1, \text{vocabulary size})$. The next token selected is the token that has the highest probability in the decoder's softmax output. Training splits into two routes at this point. We use a hyperparameter ϵ to determine the probability of feeding the decoder's output token at time t as the input token at time $t+1$ for the decoder, or to use the ground truth token at time t as the input token at time $t+1$ for the decoder. This concept is known as *teacher enforcing* [1]. This mechanism allowed the model to train faster at earlier stages of the training process when the model's output is inaccurate.

Loss and Optimizer

The loss used for this model is the cross-entropy loss because we are comparing the sequence of predicted token's probability distribution against ground truth probability distribution (1 at the correct token index). The optimizer we used is Adam [5] because as a baseline model, we wanted to have a simple stochastic optimization method that provides some degree of adaptability when estimating sparse and noisy gradients.

4.4. **Improved Model.** Our improved model consists of a multi-headed self-attention transformer architecture [13]. The architecture diagram can be found in the appendix Figure 2. Data preprocessing still follows the same process discussed in *Preprocessing* section.

Preprocessing Since the original dataset contained raw news articles with their corresponding summary taglines of varying lengths, we filtered out some of the articles that have overly long or short body text lengths or summary lengths. For this purpose, we have set a threshold for filtering articles to be at least 300 tokens long (after preprocessing). We trim off tokens exceeding 300 to ensure that each preprocessed article has exactly 300 tokens. Similarly for summaries, our threshold is set to be 40 tokens. Similarly for summaries, we filter it with a predefined threshold. This resulted in a fraction of stories remaining for training, validating and testing.

The overall process is as follow, with detailed discussion following:

- The preprocessed article body x is passed to the word embedding layer, generating a word embedding from the pre-trained word vector $emb(x)$
- $emb(x)$ is then passed through a positional encoding layer, which produces a positional embedding that is added to the word features $feat_x$
- $feat_x$ is passed through an encoder that will produce encoded features $enc(feat_x)$
- The preprocessed ground truth summaries y are passed to the decoder, along with a initial special token prepended, as a whole, along with the encoded features $enc(feat_x)$
- The decoder will output the generated summary as an entire sequence

The overall architecture of this model is comprised of two main components:

Encoder: The encoder consists of multiple attention blocks, each block is made up of a multi-head attention network and a position-wise feed-forward network. The multi-head attention network allows the encoder to compute attention scores in parallel, each head focusing on a different portion of of the input sequence, yielding d dimensional output values. The output of this network is an attention score that represents the relevancy of a particular token to the current examined token. The self-attention mechanism allows the decoder to attend to input sequences from previous positions and the current position. The actual computation can be shown below:

$$\alpha = \text{softmax}\Big(\frac{Q \cdot K^T}{\sqrt{Emb_{dim}}}\Big)$$

$\alpha -$ Similarity score for each of the keys

$Q -$ Query, a token represented by a word vector, for which we want to know its relevance within the context

$K -$ Keys is the set of tokens that we are searching from to find the candidate value

$$\text{Attention Score} = \alpha V = \sum_{i=0}^{m} \alpha_i \cdot v_i$$

$V -$ Values is also a set of tokens, in our task, is the same as keys

To prevent the attention from peaking into tokens in the future sequences, we have applied a masking operation on the input, essentially filling future positions where the input is supposed to be masked with a large negative value (i.e. -10^7). This will cause the softmax to output a near zero probability for those positions.

The position-wise feed-forward network is comprised of two feed-forward layers with a ReLU activation and a normalization layer, taking in input of shape (batch size, sequence length, feature dimension). This network allows input at every position to have a weight representing that position. This operation can also be done in parallel, even though we are working with sequential data, the network can be seen as two convolutions with kernel of 1.

Sometimes, the same tokens take on different meanings at different positions, we need to distinguish one token being at position 1 versus position 5. For this purpose, we will encode the position information in the form of positional embedding [4]. This embedding has dimension same as the pre-trained word embedding dimension. The encoder then adds this new "feature" on to the existing word embedding.

Decoder: The decoder follows a similar architecture as the encoder. With an additional attention layer for computing the attention scores for the encoder output $enc(x)$. Specifically, the multi-head self-attention network takes in input y as the query, key and value and produces att_{self}. The multi-head encoder attention network takes in input att_{self} as the query and encoder output $enc(x)$ as the key and value and produces $att_{enc-dec}$. Finally, $att_{enc-dec}$ is passed through a fully connected layer to generate logits of shape (batch size, sequence length, vocabulary length). The logits is converted into a probability distribution by applying a softmax function. Note that this softmax operation incorporates label smoothing to prevent the model from being overly confident on its predictions and overfitting [11].

Other details:

- Gradients are clipped at 1 to mitigate any gradient explosion or vanishing from affecting the training process.
- The training process employs an adaptive learning rate schedule that follows a fixed learning rate of 10^{-3} initially, and then decaying exponentially after training for a certain number of episodes

Hyperparameter Tuning

Hyperparameter	**Value**
Learning Rate	10^{-3} with adaptive exponential decay
Batch Size	32
Number of Heads for attention	10
Dropout Rate	0.2
Encoder Hidden Dimension	512
Decoder Hidden Dimension	512
# of Hidden Layers in Decoder/Encoder each	3
Key Dimension	64
Value Dimension	64

Table 1: Best Hyperparameters

5. Experimental Results

5.1. **Evaluation Metric.** The most common evaluation metric for text summarization is Recall-Oriented Understudy for Gisting Evaluation (ROGUE) [8]. ROGUE-1 recall is defined as total number of overlapping words divided by total words in the reference (ground-truth) summary. ROGUE-1 precision is defined as total number of overlapping words divided by total words in the system (generated) summary. ROUGE-1 F1 measure is the standard harmonic mean between ROUGE precision ROUGE and recall. Similarly, one can define ROUGE-2 measures by counting bigrams instead of words. One can define ROUGE-L measures by counting longest subsequence. To better present the result, we tokenize the reference and system summaries before computing ROGUE scores. We present results only for ROUGUE-1 and ROUGE-L because they are most widely used in the literature.

5.2. **Experimental Results.** Table 2 below present best results from our baseline model (model 1) and our improved model (model 2). These results are obtained by separately tuning hyperparameters on each model. Since our improved model has far better performances than the baseline, in table 2, we then present the performances on different hyperparameter combinations. Model l has relatively poor performances compared with model 2 which can be shown in detail in the comparison discussion below.

	ROUGE-1	ROUGE-L
Model 1 (Baseline model)	0.17	0.13
Model 2 (Improved model with best hyperparameters)	0.35	0.27
Model 3 (Improved model with dropout rate of 0.1)	0.3	0.25
Model 4 (Improved model with 1024 hidden layer neurons)	0.27	0.22

Table 2: ROGUE F1 scores for different models

Models 3 and 4 have similar hyperparameters as model 2 (see table 2). The only difference is that in model 3, we use a dropout rate of 0.1 and in model 4 we use hidden dimension 1024 for both encoders and decoders. In our some experiments, we find than using a lower dropout and larger hidden dimension leads to training data having a ROUGE-1 F1 score over 0.8, a clear sign of overfitting.The improvement in performances in model 2 can be attributed to the reduction in over-fitting.

Figure 3 in the appendix plots the validation and training losses for model 2. Validation loss becomes flat after roughly 20000 epochs while training loss keeps on decreasing. Note that these losses are computed after performing label smoothing, further discussion on this topic can be found in the section 4.4 Improved Model section. Meanwhile, figure 4 plots the distribution of ROUGE-1 F1 scores for on the validation set, as we can see the distribution is relatively normal, with a lightly fatter right tail. We manually inspect these surprisingly good predictions, and find that most are actually short news articles with short reference summaries.

Comparison between Model 1 and 2: Figure 5 in the appendix shows the reference summary, with system summaries from two models. Sample 1 is generated by our model 1 (baseline model), and sample 2 is generated by our model 2 (improved model). The highlighted parts are where the system summary and the reference summary overlap, so having more overlap means a better summary. The reason why model 1's summary doesn't perform well is because our baseline model relies on a recurrent network to model the input sequence, but if the input article is too long, the dependencies between the earlier tokens and the later tokens will be very small, causing the encoder to lose that information in its hidden state. Finally, the decoder will not be able to generate any summary with tokens that have occurred earlier in the input article sequence, this obstacle can be shown in the Sample 1 summary generated by model 1.

6. Discussion

There are several of issues that we came across during our preprocessing stage. Since for sequence-to-sequence models we had to add in special padding/start/end tokens to the original article and summary text, our initial tokens were $\langle START \rangle$, $\langle PAD \rangle$ and $\langle END \rangle$, which were converted to $start$, pad and end after our preprocessing stage. This caused our baseline model to not be able to learn anything because it wouldn't ignore the vast amounts of pad characters in the article and summary, causing it to output all pad tokens as the entire summary. This showed the importance of preprocessing early in the pipeline, and closely verifying all special tokens are treated specially. Another interesting finding during our experiment with the baseline model was that it always generated the same sequence over and over shown in figure 6 in the appendix. This was the motivation for using the *teaching enforcing* mechanism, it allowed the model to calibrate itself with the true summary sequence, instead of getting stuck in a saddle point.

For the improved model, the results show a significant improvement over traditional recurrent architecture in tackling sequence-dependent tasks. This result closely aligns with the theoretical foundations of the limitations of recurrent neural networks that lose valuable information in longer-term sequences, which is important in our text summarization task. The improved model's self-attention mechanism is not only efficient to compute via matrix multiplication over the entire sequence, as opposed to a serial computation of weights for every token in the sequence, but also extracts much richer information for both the encoder and decoder.

A potential area of improvement for tackling text summarization is to employ beam search into the decoding process. Since the decoder samples the token that has the highest probability out of a softmax output probability distribution. Our current approach is to always picking the max probability token for every position of the output sequence, which is very greedy in a sense. We could instead use beam search heuristics to approximate the best tokens at every position, leading to a better overall loss of the generated sequence.

Since training language models are very time consuming, we have used a pre-trained word embedding to bootstrap our two models, gaining a boost in performance. However, these pre-trained word embeddings are limited by their vocabulary size. The word embedding that we used only contained 42000 tokens in the vocabulary. Whenever our model encounters a unseen token in the test set, it would be converted into unknown word vector (with default weight of 0), essentially losing valuable information. A potential improvement would be to use a sub-word level tokenization method called byte pair encoding [3]. This tokenization method can be utilized to build a more comprehensive word embedding without dropping very rare words, because they can just be represented by special concatenations of more widely seen vocabulary. This is actually similar to how the famous language model Bidirectional Encoder Representations from Transformers (BERT) constructs its vocabulary [2].

Finally, we did not implement weight sharing between the source and target embeddings. If our task was translation, then it might not make sense to share them because the language heuristics would be different. But for text summarization, this could be a potential area to try out.

REFERENCES

[1] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1171–1179. Curran Associates, Inc., 2015.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, February 1994.

[4] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Julian Kupiec, Jan Pedersen, and Francine Chen. A trainable document summarizer, 1995.

[7] Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. Deep recurrent generative decoder for abstractive text summarization, 2017.

[8] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[9] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[11] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions, 2017.

[12] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks, 2017.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

APPENDIX

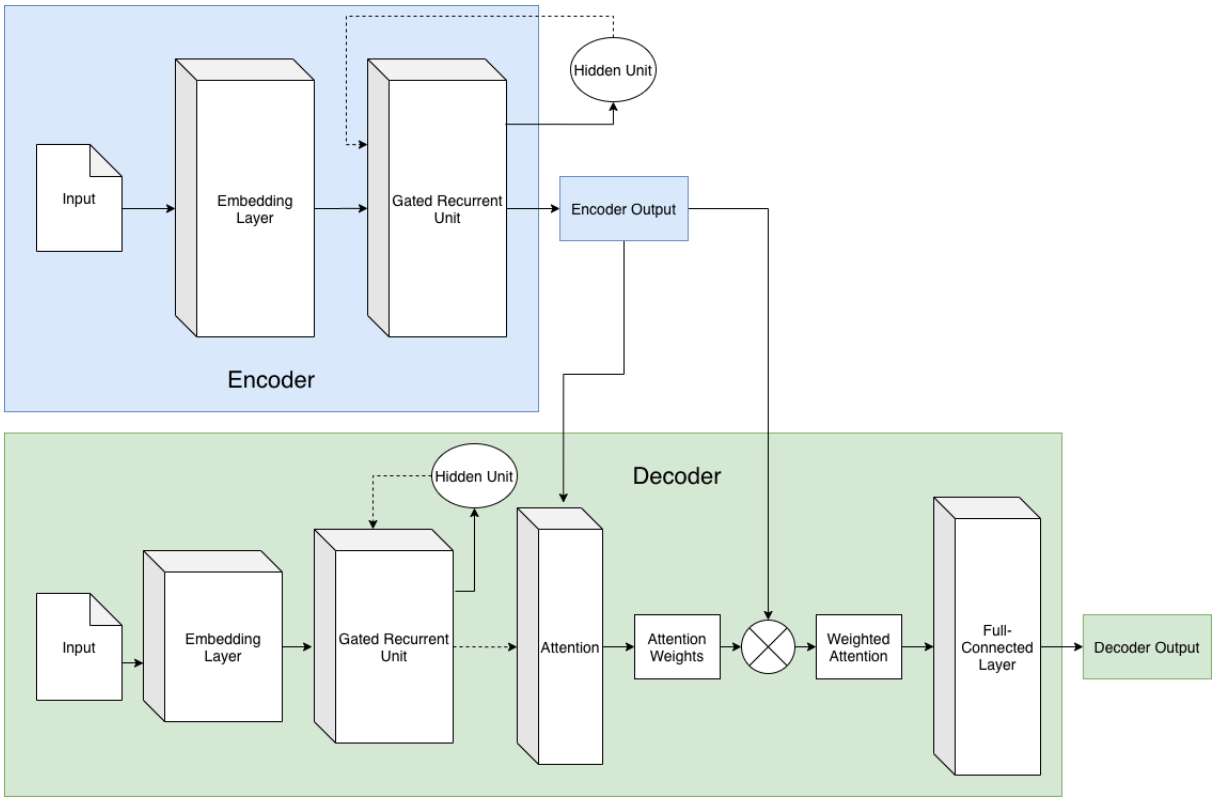

FIGURE 1. Baseline Model Architecture

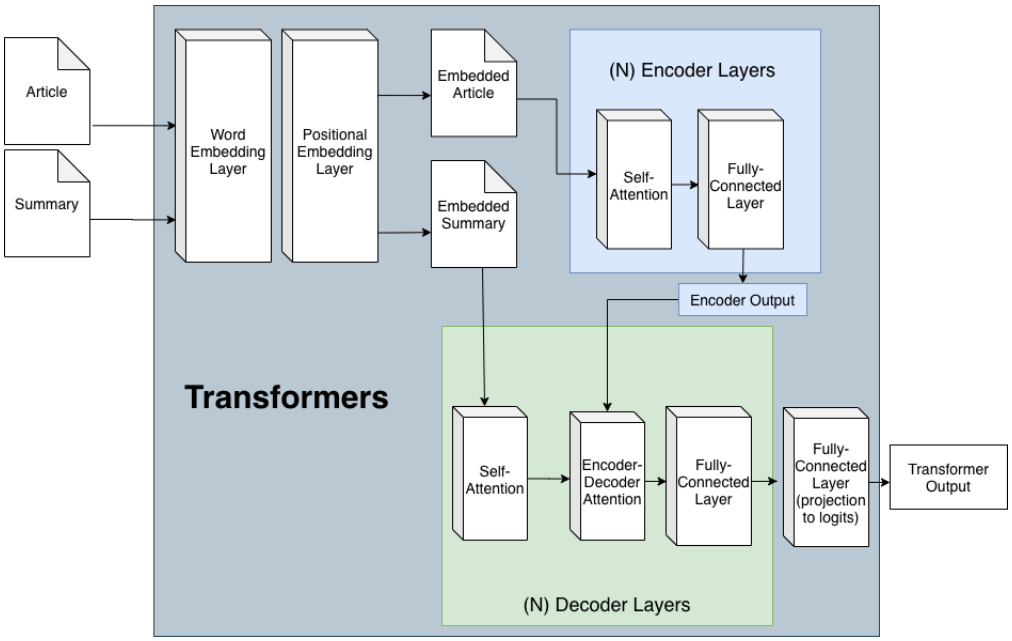

FIGURE 2. Improved Model Architecture

FIGURE 3. Training vs Validation loss on model 2

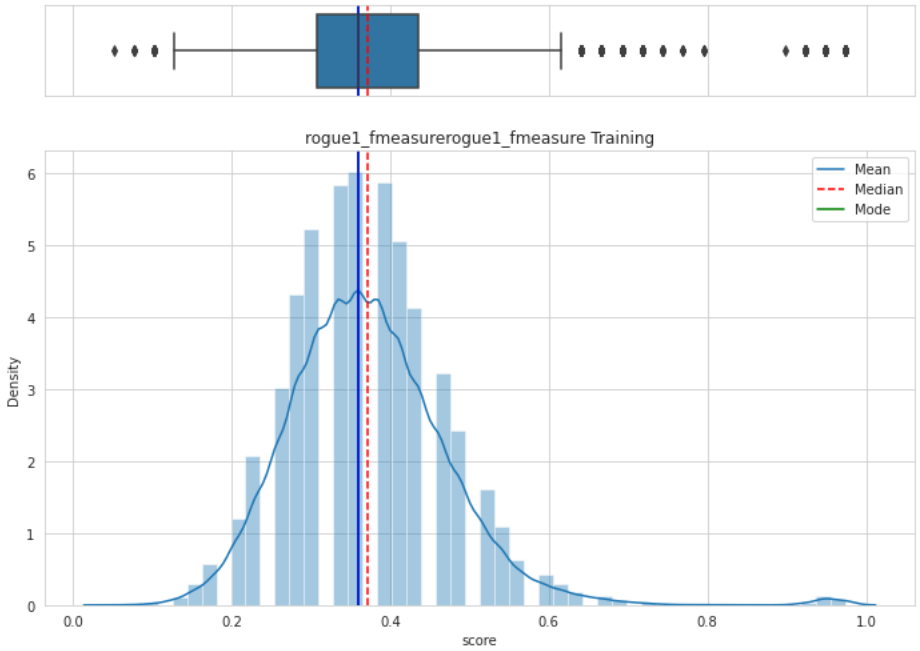

FIGURE 4. ROUGE-1 Scores

Sample 1 ROUGE-1 F1 = 0.21

SYSTEM SUMMARY

the are not the all soon christmas of our the new says the church to the jackpot into of one the church say the was was in people and the of in arkansas alabama the in says message to

REFERENCE SUMMARY

new we will accept it as the will of god victims neighbor from mennonite community says tractortrailer crossed median and struck van police say truck driver killed two children in safety seats in van survive crash ntsb sending team

Sample 2 ROUGE-1 F1 = 0.52

SYSTEM SUMMARY

shan 4yearold born in the smithsonian national zoo in boy was years gift and by ice and and and pears zoo curator says pandas are incredibly and of are young cub in they in seattle tai shan says to

REFERENCE SUMMARY

tai shan was born at the smithsonian national zoo birthday boy gets special cake made of ice beets apples and pears zoo curator says pandas are popular because they resemble young children though born in us tai shan due

FIGURE 5. Two sample system summaries

cnn one of the biggest tv events of all time is being reimagined for new audiences . roots the epic miniseries about an african american slave and his descendants had a staggering audience of over million viewers back in . now a e networks are remaking the miniseries to air in . a e lifetime and history formerly the history channel announced thursday that the three networks would simulcast a remake of the saga of kunta kinte an african who was captured shipped to america and sold into slavery to work on a virginia plantation . levar burton who portrayed

---

the year old was a strong of the murder of the murder of the year .
the year old was a strong of the murder of the murder of the year .
the year old was a strong of the murder of the murder of the year .
the year old was a strong of the murder of the murder of the year .
the year old was a strong of the murder of the murder of the year .
the year old was a strong of the murder of the murder of the year .
the year old was

FIGURE 6. Repeating summary