Transformers and Pointer-Generator Networks for Abstractive Summarization

Jon Deaton
Stanford University
jdeaton@stanford.edu

Austin Jacobs
Stanford University
ajacobs7@stanford.edu

Kathleen Kenealy
Stanford University
kkenealy@stanford.edu

Abigail See
Mentor
Stanford University
abisee@stanford.edu

Abstract

Recently, transformers have outperformed RNNs on sequence to sequence tasks like machine translation. Nima Sanjabi [15] showed that transformers also succeed in abstractive summarization tasks. However, like vanilla RNNs, transformer models produce summarizations that are very repetitive and often factually inaccurate. We improve on the transformer model by applying variants used to successfully solve these problems on RNN summarization models. We use techniques such as n-gram blocking and coverage loss to reduce repetition. We also add a pointer-generator network to allow the model to copy words from source texts and thus reduce both unknown word tokens and incorrect factual representations. We explore whether these mechanisms improve performance on the summarization task and whether they reduce redundancy and factual error as they do for RNN models. We find that pointer-generator networks improve ROUGE scores and the fluidity of summaries. Coverage and n-gram blocking also improve ROUGE scores and reduce repetition, with n-gram blocking significantly outperforming coverage.

1 Introduction

Summarization, a core problem facing modern natural language processing systems, is the task of reducing a text, such as a sentence, article, or even novel, to a shorter version that retains the quintessential ideas of the original text. Most summarization systems take one of two approaches: the *extractive* approach, whereby a system identifies and copies key phrases from the original text to form a new summary, or the *abstractive* approach, whereby a system generates entirely new words and phrases to write a summary. Extractive summarization is considered the easier task, as the copying of words and phrases tends to ensure grammatical and factual correctness. Abstractive methods, on the other hand, more accurately replicate the human summarization process, and a successful abstractive model can demonstrate the knowledge of important skills such as generalization and paraphrasing.

Before 2014, the majority of work regarding text summarization made use of the extractive approach with relative success [16]. After the development and success of sequence to sequence models pioneered by Sutskever et al. [17] in 2014, abstractive summarization based on seq2seq models (especially RNNs and LSTMs) became widespread, competing with extractive models for state-of-the-art results. Finally, in 2017 with the release of the seminal paper *Attention is all you need* [19], transformers began to surpass RNN models in terms of both performance and popularity on seq2seq tasks like machine translation. Nima Sanjabi demonstrated that tranformers also perform well on the abstractive summarization task [15].

However, both RNNs and transformers face similar issues in abstractive summarization. Both models create summaries with too much repetition, fail to handle out-of-vocabulary words, and have many factual inaccuracies. While these issues have been addressed in RNNs by the use of pointer-generator networks and coverage vectors [16], these methods have yet to be tested on transformers.

Given the promising results of transformers on abstractive summarization, the similar weaknesses of summaries produced by both RNNs and transformers, and the success of variants (such as those mentioned above) in addressing these issues in RNNs, we present new variants of the transformer that make use of pointer-generator networks, coverage vectors, and n-gram blocking to reduce the issues transformers face in abstractive summarization. We use the CNN/DailyMail dataset, as it is one of the most popular datasets for summarization and makes for easy comparison to related work. We find that the pointer-generator network improves ROUGE scores for transformer models, as do coverage and n-gram blocking.

Our pointer-generator network, based on that used by See et. al. [16], allows the transformer to point to and copy words from the source text, as well as generate new words and phrases. This technique combines the extractive and abstractive approaches in order to better handle out-of-vocabulary words and factual inaccuracies. In addition, we compare a coverage loss also as proposed by See et. al. [16] against n-gram blocking as techniques to reduce repetition in summaries. We find that n-gram blocking significantly outperforms coverage in terms of both ROUGE scores and fluidity of summary.

2 Related Work

2.1 RNNs and Related Models

Rush et. al. [13] were some of the first to use modern neural methods for abstractive summarization to high results. On both the DUC-2004 and Gigaword datasets, they found that their attention-based sumarization method scored several ROUGE points higher than the then-current state of the art systems. After the development of LSTM models by Sutskever et. al. in 2014 [17], the use of LSTMs to solve many natural language processing tasks quickly grew. Chopra et. al. [1] were some of the first to use LSTMs for abstractive summarization and improved on the scores of Rush et. al. by several ROUGE points. They additionally showed that LSTMs had significantly lower perplexity than the attention-based models used by Rush et. al.

There have been many additional improvements on this basic LSTM model for abstractive summarization, including adjusting summary style for user preferences (Fan et. al. [2]). However, the most relevant to our work is that of See et. al. [16] who tested the impact of pointer-generator networks and coverage on RNNs. See sections 2.3 and 2.4 for more detail on their work.

2.2 Transformers

Recently, a new neural architecture called a transformer has surpassed RNNs on sequence to sequence tasks like summarization in terms of both performance and popularity. The seminal paper on transformers by Vaswani et. al. [19] demonstrated that transformers produce state of the art results on machine translation, while allowing for increased parallelization and significantly reduced training time. A recent paper by Nima Sanjabi [15] also showed that transformers perform well in comparison to various seq2seq models in summarization tasks.

Otherwise, there has been little work around the use of transformers on abstractive summarization, and even less work that goes beyond simply using a basic transformer model on various datasets. One of the few papers that alters this simple model was written by Liu et. al. [6], who made use of a two-stage extractive-abstractive approach to create Wikipedia articles from several reference articles. Their abstractive stage consisted of an isolated transformer decoder used to summarize these reference articles. Their work produced state of the art results; however, their focus was on multi-document summarization and does not address the common pitfalls of transformer summarization mentioned previously.

2.3 Pointer-generator Networks

The pointer-generator was first proposed by Vinyals et. al. [20]. Their work focused on solving the convex hull, delaunay triangulation, and traveling salesman problems. The pointer-generator has since been applied to several natural language processing tasks, including translation (Gulcehre et al. [4]), language modeling (Merity et al. [7]), and summarization (See et al. [16]).

Recent work on pointer-generator networks for abstractive summarization includes that by Miao and Blunsom [9], who applied pointer-generators to sentence compression for state of the art results, and Gu [3], who similarly produced state of the art results in summarizing the LCSTS dataset. These papers largely inspired the work done by See et. al., who improved on these previous models by explicitly calculating p_{gen} and using only one attention distribution (versus Gu et. al. who use two distributions). See et. al. [16], who applied their model to an RNN, produced state of the art performance on abstractive summarization, and their model is the basis for our pointer-generator network. It should be noted that the model constructed by See et al. largely contrasts the work famously done by Nallapati et. al. [10], who only used pointer-generator networks when activated by out-of-vocabulary words.

2.4 Coverage and N-gram Blocking

Coverage was initially proposed for machine translation by Tu et. al. [18] and Mi et. al. [8] to significant success. See et. al. [16] adapts the idea of coverage to reduce repetition, which is the basis for our use of coverage. Their definition of coverage, in turn, largely derives from that used by Xu et. al. for image captioning [21].

N-gram blocking is another technique used to reduce repetition. The technique is so common to natural language processing models that it is not credited to anyone in particular.

3 Approaches

3.1 Baseline: Transformer

For our baseline, we use an open-source implementation [5] of the basic transformer architecture as laid out in the seminal paper *Attention is all you need* [19].

3.2 Coverage

3.2.1 N-Gram Blocking

We reduce repetition in summaries by adding n-gram blocking to our decoder. The decoder uses beam search to construct hypotheses and then selects the hypothesis with the largest total score. In selecting the next word for a beam, n-gram blocking simply eliminates options which would create an n-gram that already exists within the beam. For example, given the beam "*to be or not to*", 2-gram blocking would eliminate the word *be* as an option for the next word to come because the 2-gram *to be* already exists within the sentence.

3.2.2 Coverage Loss

We also use a coverage loss[1] to reduce repetition in summaries. See et al. created a coverage vector c^t which is the sum of attention $a^{t'}$ over all previous RNN time steps.

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

Using this coverage vector, they defined a *coverage loss* which penalized their model for repeatedly paying attention to the same words.

$$covloss_t = \sum_i \min(a_i^t, c_i^t)$$

We calculate the same coverage vector and coverage loss for our transformer model, where a^t is the joint attention of the last transformer layer in time step t. We add the coverage loss, weighted by a

[1] This coverage loss technique was used by See et al. to improve an RNN model; they, in turn, adapted it from Tu et. al [6].

hyperparameter λ, to the the transformer's original loss function.

$$loss_t = -\log P(w_t^*) + \lambda \sum_i \min(a_i^t, c_i^t)$$

3.3 Pointer-Generator Network

We add a pointer-generator network to our transformer to allow our model to copy words from the source text. With a learned probability p_{gen}, our model will generate a new word as normal from P_{vocab}, the softmax of the transformer output. Otherwise, the pointer-generator will utilize its joint attention to selectively point to and then copy words directly from the source text.

We borrowed this technique from See et al. who used it on their RNN summarization model. Their model calculates p_{gen} using the RNN decoder's hidden state s_t, a context vector h_t^* and decoder input x_t as follows.

$$p_{gen} = \sigma\left(w_h^\top h_t^* + w_s^\top s_t + w_x^\top x_t + b_{ptr}\right)$$

We mapped s_t, h_t^* and x_t from their RNN model to analogous values in our transformer model. We generated an attention distribution over the source, a_t, by summing across the multiple-heads of the encoder-decoder multi-head attention in the last decoder layer. The context vector h_t^* was taken as the average across the source dimension of the final encoder layer's output, weighted by the source attention distribution a_t. We took the output of the final decoder layer as analogous to the RNN-decoder hidden state, s_t. The pointer-generator network for use on a transformer can be seen in Figure 1.

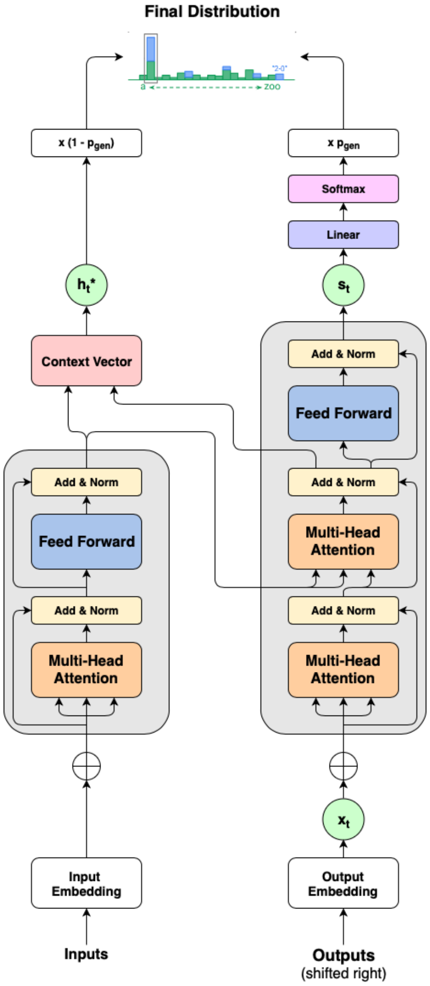

Figure 1: Pointer-generator network on a transformer; marks s_t, h_t^* and x_t for calculating p_{gen}. This original diagram is based on the diagram of a transformer presented in *Attention is all you need*[19].

4 Experiments

4.1 Data

We use the CNN/DM dataset, which consists of about $300,000$ pairs of news articles and multi-sentence summaries. However, it should be noted that the dataset is not ideal for summarization as it was originally meant for the Question Answering task. Nevertheless CNN/DM has been used widely for summarization, making it a good basis for comparison with other models.

4.2 Evaluation Method

To evaluate our results, we use ROUGE, the standard metric used for the summarization task. As the standard metric, it will allow for easy comparison across models. ROUGE is calculated as follows

$$Rouge_n(c) = \frac{\sum_{s \in S_{ref}} \sum_{gram_n \in s} match(gram_n)}{\sum_{s \in S_{RS}} \sum_{gram_n \in s} count(gram_n)}$$

where c is the candidate summary, S_{ref} is the set of reference summaries, $match(gram_n)$ is the number of times that the n-gram is found in the candidate summary, and $count(gram_n)$ is the number of times that the n-gram appears in the reference.

It should be noted that ROUGE is flawed because its focus on n-gram overlap does not handle the wide variety of valid summaries well. Nevertheless, ROUGE is the best automated metric that we have.

As we are concerned with limiting repetition and reproducing facts correctly, we will also examine the summaries by hand to determine the general performance of the model in these respects.

4.3 Experimental Details

When training our baseline, we experimented with many hyperparameters. We chose to use a single layer transformer rather than the standard 6 layer transformer. The one-layer model was complex enough to model the summarization problem and it trained much faster. We used 8 heads for the transformer's multi-headed attention, which is fairly typical. We used frozen pre-trained uncased 300-dimensional GloVe embeddings trained on 6 billion tokens with a 400k vocabulary [12].

We implemented the decaying learning rate suggested by the original transformer paper [19], which linearly increases the learning rate over a warm-up period, then decays the rate over time. During the warm-up period, however, we used a constant base learning rate because we found this custom schedule worked better empirically.

Because of time constraints, we trained our baseline and subsequent models on 40,000 examples and validated on about 5,000 examples. We used a dropout rate of 0.1, a batch size of 4, a maximum source sequence length of 400 and a maximum target sequence length of 128. Our loss was given by the cross-entropy loss for predicting the next word in the target sequence given the source sequence and all of the preceding target sequence words. We trained the baseline model for 450 epochs, which took approximately 24 hours.

We ran several training experiments on our Pointer-Generator model. We first trained the entire transformer with the Pointer-Generator from scratch and found that training took even longer than the basic transformer baseline. This was expected, given the additional parameters that must be learned by the pointer-generator. Given that the baseline transformer already took a long time to train, we then experimented with training our Pointer-Generator by restoring parameters from our baseline transformer, with and without freezing the Transformer parameters. We then trained the unfrozen Pointer-Generator model for an additional 80 epochs, which took approximately 18 hours.

For our coverage loss, we experimented with the λ hyperparameter that weights the coverage loss when it is added to the total loss. We ultimately chose to use a value of 1 for λ, which is consistent with the λ chosen by See et. al. [16]. We trained the Pointer-Generator + Coverage model for 95 epochs, which took approximately 9.5 hours.

Finally, we evaluated the Pointer-Generator model using ngram blocking where $n = 1, 2, 3, 4, 5$.

Table 1: Results

Method	R-1	R-2	R-L
Vanilla RNN (See et al., 2017)	30.49	11.17	28.08
Pointer-Generator (See et al., 2017)	36.44	15.66	33.42
Pointer-Generator + Coverage (See et al., 2017)	**39.53**	17.28	**36.38**
Transformer (Sanjabi)	27.4	-	-
Transformer Baseline	20.23	3.45	13.43
Transformer + Pointer-Generator	22.10	4.03	14.66
Transformer + Pointer-Generator + Coverage	22.93	4.08	15.08
Transformer + Pointer-Generator + N-Gram Blocking (2-gram)	25.31	4.16	15.99

4.4 Results

We have reported all of our scores for the models we trained in Table 1. First, we note that our transformer baseline has a lower score than either the Vanilla RNN or Sanjabi's transformer. We credit this difference to the fact that See et. al. trained their model for 4+ days and Sanjabi trained her model for 2+ days, while we only had the time to train our model for 24 hrs. In summary, we achieved $\frac{2}{3}$ the score of Sanjabi in half the time. We chose to prioritize testing and implementation of variants of the transformer over increasing the training time of the baseline. Given the training validation curve from our baseline, which can be seen in Figure 2, we believe that our ROUGE scores would have continued to increase if we had more training time.

As expected, we saw that the pointer-generator improved our ROUGE scores. Like the basic transformer, we saw that our ROUGE scores steadily increased over time, and if we had more time to train, we believe that our ROUGE scores would have continued to improve. The scores reported above came from restoring our baseline transformer and training for additional epochs with the pointer-generator model. A noteworthy takeaway from our experiments is that allowing the model to learn both transformer and pointer-generator parameters produced significantly higher results than freezing the transformer parameters learned during baseline training and only learning the pointer-generator parameters.

As we expected, we again improved our scores by adding coverage loss to our transformer-pointer-generator model. The scores reported above came from restoring our pointer-generator model and training for additional epochs with the coverage loss.

Similarly, we saw that n-gram blocking significantly improved our scores. Figure 3 shows ROUGE scores as a function of n for the Transformer + Pointer-Generator model. In addition, it was far more effective in regards to improving ROUGE scores than coverage. This was as we expected; coverage loss requires the model to learn how to not repeat, while n-gram blocking simply blocks all repetitive phrases (there is no learning involved; we can explicitly reduce repetition). See Section 5 for further comparison and discussion of these results.

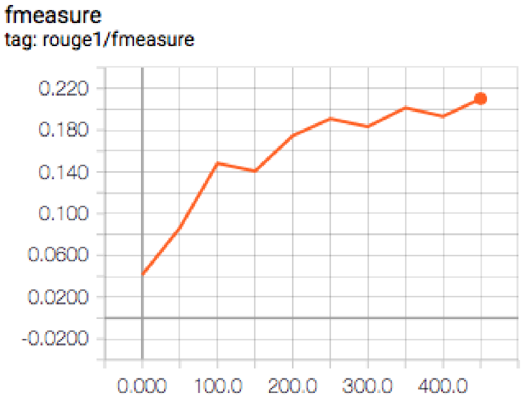

Figure 2: The ROUGE-1 validation scores for the Baseline model as a function of the number of batches trained on. It is clear that the model's scores were still improving when we stopped at 450 epochs.

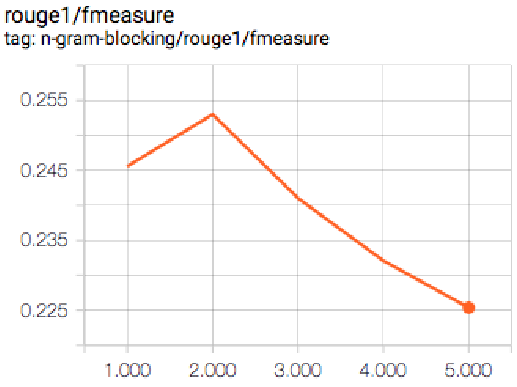

Figure 3: The ROUGE-1 scores on the test set for the Transformer + Pointer-Generator model as a function of n, the length of the n-gram used for n-gram blocking. ROUGE-1 scores are listed as a fraction of 100.

5 Analysis

Though our baseline transformer produced lower scores than expected, it did produce relatively high results given our limited training time. That being said, we can see from the example below that our summaries face significant issues; in particular, repetition is a clear issue in the example below, as is the mishandling of out of vocabulary (OOV) words, evident in the use of the "[UNK]" token.

Human Summary: oleg kalashnikov died of gunshot wounds , ukraine 's interior ministry said . he was a party of regions deputy in ukraine 's previous parliament . kalashnikov was ally of deposed ukrainian president viktor yanukovych .
Baseline Transformer Summary: viktor [UNK] was found dead at the vladimir putin 's death . he was found dead in his remained remained remained remained remained ...
Pointer-Generator Summary: russian oleg oleg oleg oleg sergei yanukovych was found dead . viktor was found dead at home in kiev in kiev . viktor yanukovych was found dead in the the kremlin has been been been been since .
Pointer-Generator + Coverage Summary: ukrainian prime minister yanukovych was found dead in his home in kiev . ukrainian leader was found dead in his home in kiev , in kiev .
Pointer-Generator + 2-Gram Summary: oleg viktor yanukovych was found dead in his home in kiev . he is of the ukraine of ukraine 's parliament . the former ukrainian president vladimir putin says he was “ “ the communist party ”.

Table 2: The human, baseline, pointer-generator, coverage, and n-gram blocking summaries produced for a single article.

Our pointer-generator network improved our ROUGE scores from our baseline; further analysis of the summaries outputted by this model show that while we successfully reduced OOV mishandling, we still suffer from factual inaccuracies. For instance, in the example below, it is clear that we no longer run into out of vocabulary words; indeed, the model uses words like "yanukovych", "sergei", and "oleg" in its summary, despite the fact that these are uncommon words that our baseline transformer could not generate. However, our model also falsely states that the Yanukovych has died, instead of stating that Oleg Kalashnikov has died. Finally, we can see that the pointer-generator has significantly improved the fluidity of our summaries by allowing us to point to words. Yet, we have not entirely addressed the factual inaccuracies that arise in abstractive summarization and still face clear repetition issues.

When adding coverage to our model, we once again improved our ROUGE scores and clearly addressed some of our repetition issues. While we no longer repeat a single word over and over, we find that our examples now begin to repeat ideas and phrases ("in kiev" and "was found dead" are repeated several times in our summary).

In comparison, n-gram blocking does not repeat phrases or single words multiple times in a row. One caveat of the n-gram blocking performance is that we get several out of place punctuation marks (in the above example, one set of quotation marks); this is probably due to the fact that placing these punctuation marks allows us to avoid repeating phrases. That being said, the n-gram blocking method clearly produces the more fluid summary and contains more unique ideas.

While we saw that n-gram blocking produced less repetitive summaries than coverage loss, we note that n-gram blocking is a much hackier way of producing summaries. It does not reflect any learning on the part of the model, whereas the improvements we saw from coverage loss do. In the larger scheme, we believe that coverage loss is a better way to train models to repeat less, while n-gram blocking can be used to make up for occasional mistakes during evaluation. Though, from our results, more work is needed to improve the effectiveness of coverage loss in reducing repetition.

6 Conclusion

We have presented new transformer models that make use of pointer-generator networks, coverage, and n-gram blocking to reduce repetition, factual inaccuracies, and OOV mishandling in abstractive summarization. We have shown that each of these models [in this order] improves on the performance of the previous models.

Our work is limited in that we did not let any of our models train to full convergence, which makes our results less comparable to those of state-of-the-art examples. Future work should include additional hyper-parameter tuning and extended training time for each model, in order to make claims about these models compared to current state-of-the-art results.

In examining the output of each of our models, we noted that the pointer-generator network eliminated OOV words, but its summaries still suffered from factual inaccuracy. We also found that n-gram blocking performed better than coverage loss for reducing repetition.

Future work could also include experimenting with model architecture, including additional Transformer layers and different mechanisms of calculating the attention distribution over the source sequence. It would also be interesting to experiment with applications to other data sets including the New York Times Annotated Corpus [14] and Gigaword [11].

References

[1] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. *North American Chapter of the Association for Computational Linguistics*, 2016.

[2] Angela Fan, David Grangier, and Michael Auli. Controllable abstractive summarization. 2018.

[3] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporating copying mechanism in sequence-to-sequence learning. *Association for Computational Linguistics*, 2016.

[4] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *Association for Computational Linguistics*, 2016.

[5] Yu-Hsiang Huang. Attention is all you need - pytorch. `https://github.com/jadore801120/attention-is-all-you-need-pytorch`, 2018.

[6] Peter J. Liu and Mohammad Saleh. Generating wikipedia by summarizing long sequences. *International Conference on Learning Representations*, 2018.

[7] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *NIPS 2016 Workshop on Multi-class and Multi-label Learning in Extremely Large Label Spaces*, 2016.

[8] Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. Coverage embedding models for neural machine translation. *Empirical Methods in Natural Language Processing*, 2016.

[9] Yishu Miao and Phil Blunsom. Language as a latent variable: Discrete generative models for sentence compression. *Empirical Methods in Natural Language Processings*, 2016.

[10] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. *Computational Natural Language Learning*, 2016.

[11] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition. *Linguistic Data Consortium*, 2011.

[12] 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.

[13] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *Empirical Methods in Natural Language Processing*, 2015.

[14] Evan Sandhaus. The new york times annotated corpus. *Linguistic Data Consortium*, 2008.

[15] Nima Sanjabi. Abstractive text summarization with attention-based mechanism. Master's thesis, Universitat Politècnica de Catalunya, July 2018.

[16] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 1073–1083, July 2017.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Neural Information Processing Systems*, 2014.

[18] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. *Association for Computational Linguistics*, 2016.

[19] Shazeer N. Parmar N. Uszkoreit J. Jones L. Gomez A. N. Kaiser L. Vaswani, A. and I. Polosukhin. Attention is all you need. *In the Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[20] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Neural Information Processing Systems*, 2015.

[21] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning*, 2015.