Discriminative Adversarial Search for Natural Language Generation

Abhik Singla `abhiksingla` **Harsh Malara** `hmalara` **Sreekar Garlapati** `sgarlapati` **Videsh Suman** `vsuman`

1 Problem statement

The goal of this project is to reproduce the work on "Discriminative Adversarial Search for Abstractive Summarization" (Scialom et al., 2020), thus reproducing the results and investigating efficacy of the approach. In this paper, the authors proposed a novel method for natural language generation, Discriminative Adversarial Search (DAS), which claims to alleviate the effects of exposure bias without requiring external metrics like BLEU or ROUGE. The work demonstrated results on the task of Abstractive Summarization, improving over the state-of-the-art methods. Further, they showed that DAS can be useful for cross-domain adaptation. Their final claim is that DAS does not rely on additional rule-based filtering strategies for generated outputs, commonly used by the state-of-the-art Natural Language Generation (NLG) systems.

We feel that the claims of this paper are certainly interesting, which motivates us to pursue this direction of investigating the reproducibility and reliability of their method and experiments. As much as possible, we intend to follow the reproducibility guidelines here[1], and concurrently submit our findings to the ML Reproducibility Challenge 2020. If time permits, we would like to report results for NLG and cross-domain adaptation on one other task (like question answering), besides abstractive summarization.

We hope to be able to contribute to the open source community and contribute in the effort to cross-check published results while learning the standards and practices involved in reproducibility checking of modern Computer Science research.

[1] `https://paperswithcode.com/rc2020`

2 What you proposed vs. what you accomplished

- ~~Benchmark and fine-tune `UniLM` on CNN/DM dataset.~~
- ~~Design and train a discriminator model and modify beam search.~~
- ~~Develop DAS architecture.~~
- ~~Benchmark DAS-single architecture.~~
- *Benchmark DAS-retrain architecture* We were unable to do this because the our observed time and compute estimates to run this is significantly larger than that mentioned in the paper. Also, we started two weeks later as planned.
- *Perform cross-domain performance analysis.* We failed to do this because prior components took longer time.
- ~~ML reproduciblility best practices checklist~~
- ~~Work on final reports and code freezing.~~

The code is available at https://github.com/gitting-it-right/das_nlp.git

3 Related work

A majority of literature on NLG consists of sequence-to-sequence models trained via maximum likelihood estimation. At inference time, such a model has only access to its previous outputs, and usually suffers from *exposure bias* (Bengio et al., 2015; Ranzato et al., 2015): accumulation of mistakes resulting in a divergence from the training set distribution, and thus, poor generation outputs. Works by Wiseman and Rush (2016) and Paulus et al. (2017) propose to optimize sequence level metrics like BLEU or ROUGE to tackle this issue

Total	Train	Validation	Test
311,971	287,113	13,368	11,490

Table 1: CNN/Daily Mail (CNN/DM) dataset distribution

for NLG. However, these metrics do not reflect meaning preservation, nor do they map so well to human judgements.

GANs (Goodfellow et al., 2014) represent a natural alternative to learn to generate text that a discriminator cannot differentiate from the human-produced content. However, text GANs have failed to produce state-of-the-art results mainly due to the non-differentiable classifier signal (due to discreteness of textual data) and mode collapse. Previous works tackled various NLG tasks using text classifier as discriminators. Chen et al. (2020) leveraged on discriminators to improve unconditional text generation following Gabriel et al. (2019) worked on summarization. The work by Chen et al. (2020) is the closest work to the proposed approach.

Beam search is a widely accepted algorithm used to decode generated sequences of text, in NLG taks. This decoding strategy offers more flexibility than a greedy approach and allows to select the sequence with the highest probability. Gehrmann et al. (2018) added external rules like inclusion of a length penalty to further constrain the generation Hokamp and Liu (2017) reported improvements when adding lexical constraints to beam search. Paulus et al. (2017) introduced a rule in the beam forbidding the repetition of 3-grams to reduce repetitions.

Current state-of-the-art results either trained from scratch Gehrmann et al. (2018) or based on pre-trained language models Dong et al. (2019), have benefitted using constraints like length penalty and 3-grams repetition avoidance.

4 Dataset

We focused on CNN/Daily Mail (CNN/DM) summarization dataset[2] Nallapati et al. (2016); Hermann et al. (2015) The dataset consists of news articles paired to multi-sentence summaries. The summaries were written by professional writers and consist of several bullet points corresponding to the important information present in the paired articles. The source articles and target summaries have an average token length of 810.69 and 61.04 respectively. The distribution of the CNN/DM dataset in summarized in Table 1

[2]CNN/Daily Mail (CNN/DM) Dataset: `https://github.com/becxer/cnn-dailymail/`

4.1 Creating Dataset for Discriminator Training

We are required to generate a discriminator model as discussed in more detail in Section 7. The goal of the discriminator model is to classify a given sequence or summary as human-written (H) or machine-produced (G). We created our own dataset to train the discriminator model using the CNN/DM dataset. We sampled $150,000$ article summary pairs from the CNN/DM dataset and labelled these summaries as human-written. Next, we generated summaries of these articles using `UniLM` Dong et al. (2019) generator fine-tuned on CNN/DM and labelled them machine-produced.

Now, we have a total of $300,000$ summaries divided equally among the human-written and machine-produced labels. The discriminator model also needs to learn on the partial summaries hence each sequence is further divided to partial sequences (upto t tokens) while retaining the same label as shown in equation 1.

$$\begin{aligned} H &= \{(x, y_{1:t}) | x \in X \wedge y \in H(x) \wedge t \leq |y|\} \\ G &= \{(x, y_{1:t}) | x \in X \wedge y \in G(x) \wedge t \leq |y|\} \end{aligned} \tag{1}$$

For example: a summary sequence like “Ken eats apple” is further divided into three sub-sequences as “Ken”, “Ken eats”, “Ken eats apple” and finally used for training. The maximum number of timesteps (t) taken is 140, therefore, the final size of the dataset generated is of the order 40 million sequence label pairs.

The last pre-processing step before a sequence label pairs can be fed into the model is pre-pended the sequence with the [CLS] token, inserting a [SEP] between the article and the summary and placing a [SEP] at the end of the summary. The output after the previous step can be sent directly to the neural model as input.

5 Baselines

The paper is build upon the Unified Language Model (UniLM) for natural language understanding and generation (NLG/NLU) proposed by Dong et al. (2019). It is the state-of-the art model for summarization which was the reason, the authors

Data Points	Beam Size	Time Taken
10,000	1	1 hrs 58 mins
10,000	5	8 hrs 43 mins

Table 2: Generation time at different beam sizes

chose it as the base model for the generation (decoding) task. This model can be described as a Transformer (Vaswani et al., 2017) whose weights are first initialised from BERT. However, BERT is an encoder trained with bi-directional self attention: it can be used in Natural Language Understanding (NLU) tasks but not directly for generation (NLG). Dong et al. (2019) proposed to unify it for NLU and NLG: resuming its training, this time with an unidirectional loss; after this step, the model can be directly fine-tuned on any NLG task.

For generation of our abstractive summaries we used a pre-trained encoder-decoder structured UniLM model for the natural language generation task (NLG). Beam search is the de-facto algorithm used to decode generated sequences of text, in NLG task. This decoding strategy allows to select the sequence with the highest probability, offering more flexibility than a greedy approach. The decoding in this summary generation is based on beam-search algorithm. We have used beam search with both 1 and 5 as beam size for the summary generation.

Implementation details: As a pre-processing step a [CLS] token is added before the article tokens, and [X_SEP] at the end of each article sentence. There is a [SEP] token input at the first time-step of the decoder. Padding is done to perform the decoding in batches. Another optimisation in the code is rearranging the articles according to the their lengths to reduce the batch token length, thereby reducing the [PAD] tokens required. The baseline model also clips the input article token sequence length to 768 and limits the output target token sequence length as 128.

The baseline model itself took a lot of effort to run on Google Colab, which was really helpful in debugging and fixing a lot of environment related issues later in the experiments. The decoding on the dataset was done on the Gypsum GPU because of the high decoding time requirements as shown in Table 2.

6 Implementation Challenges

For the implementation of this project all the group members had Google Colab access and one of us had gypsum access with limited number of GPU's. A `UniLM` based model is utilized for both the generator and the discriminator, which itself uses either bert-base-cased or bert-large-cased (based on the user requirement).

The repository containing the `UniLM` code has specific versions requirement for both the CUDA libraries (eg. cudnn and apex library) and other standard machine learning python packages. We found that the current UniLM code is incompatible with the latest version of these standard libraries. This caused us to spend a significant time in changing the default GPU libraries and the python packages versions that are already present in the Google Colab environment. Running the Colab notebook was really important as we needed to debug the different components simultaneously among our team members, as only one of us had the Gypsum GPU access. The training and evaluation for all the sub-components was done by the member with the gypsum access which turned out to be a un-resolvable bottleneck.

The paper suggested the training time using a single RTX2080 GPU to be really low in order of 300-500 mins for training the sub-components. This made us spend a lot of time to try training our sub-components in the Colab notebook, varying the batch size , dataset size and changing other variables. We faced cuda out of memory issues, and were timed-out by Colab for upto 24 hrs. This made us decide to fix the beam size to 1 for all the subsequent decoding models used by us.

On the brighter side, it gave us a really good understanding of the importance of the GPU libraries like apex, cuda etc. This turned out to be really helpful when we had to modify the beam search of the decoder, where the modified code required understanding of the apex library.

Finally, we were able to run the different sub-models in the Colab notebook so as to be able to debug the components in parallel. The environment for the Gypsum GPU was created using the conda environment which helped us to run the UniLM based sub components with relative ease.

7 Our approach

BERT-gen : The model used in the paper's ablation experiments was BERT-gen. This model is obtained by fine-tuning BERT-base on CNN/DM in the same way as UniLM without performing

Algorithm 1 DAS: a Beam Search algorithm with the proposed discriminator re-ranking mechanism highligted.

```
Require: $B$, $T$, $K_{rerank}$, $\alpha$
1: $C \leftarrow$ {Start-Of-Sentence}
2: for $t = 1, ..., T$ do
3:     $C \leftarrow \{\hat{y} | (\hat{y}_{1:t-1} \in C \wedge \hat{y}_t \in V) \vee (\hat{y} \in C \wedge \hat{y}_{-1} = \$)\}$
       # Pre-filter $K_{rerank}$ sequences with top $S_{gen}$
4:     $C \leftarrow \underset{\tilde{C} \subseteq C, |\tilde{C}| = K_{rerank}}{\arg\max} \sum_{\hat{y} \in \tilde{C}} S_{gen}(\hat{y})$
       # Filter $B$ sequences with top $S_{DAS}$
5:     $C \leftarrow \underset{\tilde{C} \subseteq C, |\tilde{C}| = B}{\arg\max} \sum_{\hat{y} \in \tilde{C}} S_{DAS}(\hat{y})$
6:     if only ended sequences in $C$ then
7:         return $C$
8:     end if
9: end for
```

the multi-objective pre-training described in the UniLM paper.

Discriminator Architecture : The discriminator used throughout all experiments in the paper was trained by starting from BERT-gen and modifying the architecture to perform sequence classification instead of the masked LM objective. We started from the BERT-gen model, removed the decoder layer and added a classifier on the CLS token to achieve the functionality. We defined this model as BertForSummaryDiscrimination in modeling.py.

DAS algorithm: The architecture proposed by the author involves a discriminator network which labels an input sequence as being either *human-produced* or *machine-generated*. This architecture is inspired by the GAN architecture but differs from it because at each step instead of having the discriminator predict the next word from the vocabulary, we compute the probability that the input summary was generated by a human. The probabilities of the generator's outputs according to the discriminator are used in their reranking, in the reranking step of beam search. Algorithm 1 shown above describes the steps of the DAS algorithm. The highlighted portion is the step where it differs from the regular beam search algorithm. The scores $S_{gen}(\hat{y})$ and $S_{DAS}(\hat{y})$ are computed using

$$S_{gen}(\hat{y}) = \log P_\gamma(y_{1:t-1}|x) + \log P_\gamma(y_t|x, y_{1:t-1})$$

$$S_{DAS}(\hat{y}) = S_{gen}(\hat{y}) + \alpha \times S_{dis}(\hat{y})$$

where $S_{dis}(\hat{y}) = log(D_\delta(x, \hat{y})$.

Implementation details: We understood the beam search algorithm for decoding a single and batch of sentence, however, the implementation gets challenging while decoding with discriminator for a batch of sequence. The official beam search code supporting the UniLM model was quite complicated to understand. We need to put extra attention on the tensor sizes to understand the flow in beam search for a batch of sentences.

Initially we implemented a simple pythonic (using for loops) code while modifying the beam search code for DAS approach, however, we realized that such an implementation will not scale for larger datasets as it was taking significantly high amount of time to decode. Once we finalized the algorithmic logic, we started modifying the code to run in parallel. To achieve complete parallelization, we ported the variables in tensors and carefully implemented the flow of these tensors during the beam search.

Another big challenge was to modify the beam search algorithm and integrate a discriminator model in the update rule. We need to specially take care of the format and dimensions of the data being sent to the discriminator model. We need to make copies of the already decoded output and carefully combine the new potential tokens during the reranking step. Additionally, we need to add extra [CLS], [X_SEP] and [PAD] tokens as required by the discriminator configuration (discussed earlier).

While generating the summaries for the test articles using the modified beam search, we observe that we need to make $(beam\_size \times K\_rerank \times summary\_length)$ number of inference steps through the discriminator for a single input article. Assuming $beam\_size = 2$, $K\_rerank = 5$, $summary\_length = 80$, we need about 800 discriminator passes where each pass takes a non-trivial amount of time. Thus, we expect the summary generation to take a significant amount of time even for the test articles. Considering such huge computational bottleneck, we decided to run all our experiments using $beam\_size = 1$ and $K\_rerank = 5$.

Data Points	K_{rerank}	Time Taken
10,000	5	6 hrs 42 mins

Table 3: Generation time with modified beam search (with beam size = 1)

8 ML Code Completeness Checklist

Over the timeline of our project, we tried to follow the best software and applied machine learning practises recommended by the community and thus we have tried to cover all the following paradigms for a good code reproduction, with best of our abilities.

- **Specification of dependencies :** We have created a requirements.txt file which will correspond to the python pip packages that needs to be installed. We also created a conda environment added a environment.yml file to able to make the establishing of the conda environment for users easier.

- **Training Code :** The training code is provided in form of a directory structure having a hierarchy for the internal dependency files. The discriminator, generator, modified generator (with re-ranking step) code can be executed using shell script provided. The scripts to run the various models are explained in the ReadMe file.

  The code directory structure is the one of original UniLM model, with the sub-component additions to it corresponding to the following:

  - Discriminator dataset generation script - located in src/create_das_dataset.py
  - Discriminator model file (with example scripts associated with it required for training). - located in src/pytorch_pretrained_bert/modeling.py as BertForSummaryDiscrimination
  - Modified generator model file with the modified beam search with re-ranking (which internally invokes the discriminator model in evaluation mode). - located in src/pytorch_pretrained_bert/modeling.py as forward pass of BertForSeq2SeqDecoder

- **Evaluation code :** The author has introduced some custom evaluation metrics other than the ROUGE scores. We implemented these metrics from scratch and added to the directory as ‘eval’ script which takes in the article and the summary as parameters and outputs all the required evaluation metrics.

- **Pre-Trained Models :** The Pre-trained models required anywhere in the code are directly accessible from the HuggingFace server. Additionally, we have also uploaded all the models trained at our end (for example: discriminator model, bert-base generator fine-tuned on CNN/DM dataset)

- **ReadMe File:** The ReadMe file contains the detailed instruction on setting up the project and alongwith the commands to install the required dependencies. We also included the scripts required to the run the experiments using different sub-components in different paradigms. These will include the shell scripts required to run the training or evaluation experiment. We also included a small description for each different script.

9 Results and Observations

In this section we will discuss about multiple quantitative and qualitative measures to evaluate the proposed approach. Table 5 depicts a sample summaries generated using following methods for qualitative analysis:

- the ground truth summaries
- decoding with high beam search (beam search = 5)
- decoding with DAS-single (involving single beam, and K-reranking as 5)

We observed that our model with the DAS-single is able to perform good summarization compared to the decoding using beam search 5 approach. We also observed that as the beam size increases the summaries get shorter and more generic, while the smaller beam size summaries tend to have repetition. We speculate that using the discriminator helps in reducing this problem of repetition as the discriminator model improves the re-ranking and beam search.

We also observed that the human written summaries are in active speech, while the generator decoded summaries either using beam search or the DAS approach are in passive speech. It makes

	UniLM (Beam search = 1)	**UniLM (Beam search = 5)**	**DAS - single (Beam search = 1)**
Rouge-1	41.6	41.3	41.65
Rouge-2	19.14	19.45	19.15
Rouge-L	34.75	34.70	34.77
len	-30.80	-30.46	-30.8
nov-1	11.70	12.93	11.72
nov-3	33.23	43.28	33.22
rep-1	-3.72	-1.38	-3.71
rep-3	-1.758	0.03	-1.756

Table 4: Results on CNN/DM test set for the UniLM and DAS approach

Article	Michele Bachmann is comparing President Obama to the co-pilot of the doomed Germanwings flight. ”With his Iran deal, Barack Obama is for the 300 million souls of the United States what Andreas Lubitz was for the 150 souls on the German Wings flight - a deranged pilot flying his entire nation into the rocks, ”the Minnesota Republican and former representative wrote in a Facebook comment posted March 31 ...
Ground truth	Former GOP representative compares President Obama to Andreas Lubitz. Bachmann said with possible Iran deal , Obama will fly ”entire nation into the rocks” Reaction on social media? She was blasted by Facebook commenters.
Beam search = 5	Michele Bachmann compared President Obama to co-pilot of Germanwings Flight 9525. Many comments posted on her Facebook page blasted the former representative .
DAS	Michele Bachmann compared Obama to co-pilot Andreas Lubitz. Many comments on her Facebook page blasted the former representative. She personally told Obama to ”bomb Iran” during the 2014 White House Christmas Party.

Table 5: An example of an news article and it’s summaries generated using different methods

sense that the humans after reading the article which is usually in active speech, make summaries in active speech too.

The discriminator was trained on the dataset created for the discriminator using the CNN/DM (as discussed above). The model is tuned with validation set as 20% and train set as 80% of the data. The final model checkpoint chosen (for the DAS-algorithm) as the discriminator is the one trained till Epoch = 3. The validation accuracy achieved is 89.92% (comparing with the 93% accuracy of the discriminator in the original paper). The training and validation loss over multiple epochs are shown in the graphs below. (Figure 1 and Figure 2)

Data Points	Time Taken
125000	11 hrs 12 mins

Table 6: Training Time for Discriminator, Number of Epochs = 9

9.1 Metric Definitions

We used `UniLM` Dong et al. (2019), with and without the additional constraints (like length-penalty, forbid duplicate ngrams and forbid ignore word) for reducing repetitions of ngrams, as the baselines for the DAS model. We compare the different approaches using standard evaluation metrics like ROGUE-1, ROUGE-2 and ROGUE-L See et al. (2017), Gehrmann et al. (2018).

The authors also defined the following custom metrics to evaluate the performance of the approach:

- **Novelty (nov-n)** is defined as the percentage of novel n-grams w.r.t. the source article, indicating the abstractiveness of the generated summary.
- **Repetition (rep-n)** is defined as the percentage of n-grams that occur more than once in

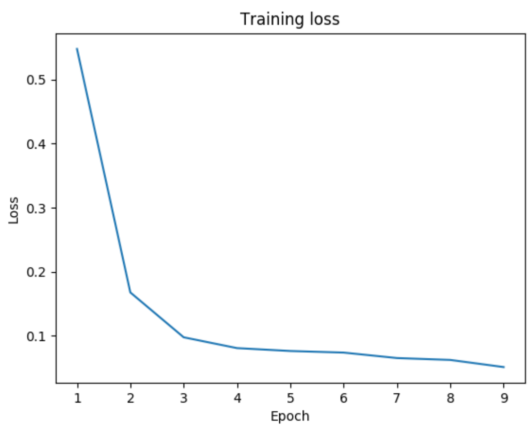

Figure 1: Training loss for discriminator

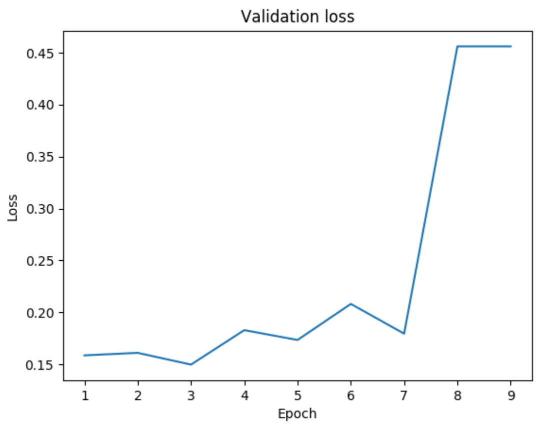

Figure 2: Validation loss for discriminator

the summary.

- **Length (len)** is defined as the length in tokens of the summary. the abstractiveness of a system

Finally, we utilize Δm of these measures to evaluate the approach where, Δm is the difference of measure m w.r.t. the human-quality summaries. It is important to note that the objective is not to maximize those metrics, but to minimize the difference w.r.t. human quality summaries. Hence, we report this difference such that for any measure m above, $\Delta m = m_{human} - m_{model}$.

9.2 Error Analysis

We talked about the training time for the discriminator, time required for the normal beam search decoding and the time required by the modified beam search decoding (involving discriminator). (Refer Table 2 and Table 3)

All the time required for the above processes above, is way more than the average 300-500 mins that the author proposed for the discriminator training using a single GPU. (Even our decoding steps usually took more or comparable time after training using 4 2080Ti GPU's). We are not able to understand why is that the case and are in conversation with the author's over email for the same.

We have not been able to run the DAS-retrain model and the domain adaptation to TL;DR. The primary reason was that our experiments were computationally expensive and would take a lot of time thereby causing a lot of time loss if we find and debug problems in our experiment results. This caused us to postpone the timeline for our subsequent experiments. We also started working on the project 2 weeks later than our timeline which caused further delay. Also our estimation of the errors and time required to understand the UniLM code was misjudged. The timeline we had planned was based on the training and decoding estimates from the paper, which turned out to be significantly different for us during the implementation phase resulting in a lot of experiments in being pushed forward.

Table. 4 reflects the quantitative measures of our experiments. We observed that the DAS approach with beam search = 1 is performing slightly better than UniLM with beam search = 1. We were unable to run DAS with beam search = 5 due to computational bottlenecks as discussed before. All the ablation results from the paper have not been replicated properly as our training and evaluation sets have been way smaller that that of the author. This has been primarily due to computation issues as discussed earlier. The DAS-single evaluation results tend to have the most difference in replication, which we speculate is due to the fact that the effect of the discriminator probabilities (if improper) accumulates over each time-step of the modified beam search which heavily affect the output generated summary, thereby impacting the scores.

10 Contributions of group members

We really enjoyed working on this project in the group setting. We realized the importance and ease of meeting and discussing issues in-person but overall we all managed to work effectively via zoom conferences. Since, all the group members were new to latest NLP tools and libraries, we realized working in pairs helped us a lot. This setting helped us to understand topics, identify the issues and implement codes efficiently.

- Abhik & Harsh:
  - Initial experiment setup in Colab, Setup UniLM and other configurations for gypsum, colab.
  - Understanding and modifying Beam search algorithm. Implemented evaluation code.
- Sreekar & Videsh:
  - Implemented discriminator data generation and discriminator training codes.
  - Fine-tuned generator on CNN/DM dataset.
- All members:
  - Integrating beam search and discriminator model.
  - Report Writing.

11 Conclusion

This project helped us a lot to deep dive into the granularity of the NLP domain code. We found that understanding of the implementation at such a granular level is really tough. During the experiments we also realised being able to get a judgement of the intermediate results in the model can really help in judging if there is something wrong in the model before we do the complete end-to-end evaluation.

We have decided to keep on working on the code base and resolve the coding and training time related issues we are facing with the help of the authors (with whom we are in contact). We plan to keep on working to add the DAS-retain component and try experiments with domain adaptation, so that we can submit our entry for the ML Reproducibility Challenge.

References

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Chen, X., Cai, P., Jin, P., Wang, H., Dai, X., and Chen, J. (2020). A discriminator improves unconditional text generation without updating the generator. *arXiv preprint arXiv:2004.02135*.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H.-W. (2019). Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pages 13042–13054.

Gabriel, S., Bosselut, A., Holtzman, A., Lo, K., Celikyilmaz, A., and Choi, Y. (2019). Cooperative generator-discriminator networks for abstractive summarization with narrative flow. *arXiv preprint arXiv:1907.01272*.

Gehrmann, S., Deng, Y., and Rush, A. M. (2018). Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1693–1701. Curran Associates, Inc.

Hokamp, C. and Liu, Q. (2017). Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546.

Nallapati, R., Xiang, B., and Zhou, B. (2016). Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023.

Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2015). Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Scialom, T., Dray, P.-A., Lamprier, S., Piwowarski, B., and Staiano, J. (2020). Discriminative adversarial search for abstractive summarization. In *Proceedings of Machine Learning and Systems 2020*, pages 7603–7612.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

Wiseman, S. and Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306.