Semantic Evaluation of Summaries generated using RNNs and GANs

Aishwarya Sudhakar Hrishikesh Kashyap
University of Massachusetts, Amherst
{asudhakar, hkashyap}@cs.umass.edu

Abstract

In this project we build two abstractive text summaries using two advanced neural network models and compare their performance using a semantic approach in addition to traditional ROUGE scores used for evaluating automatic summarization tasks. Our first neural network is a Seq2Seq RNN with a Pointer-Generator Architecture. Our second neural network is a GAN with a bi-directional LSTM as a Generator and a binary classifier as the Discriminator, to distinguish between automated summaries and human summaries to better the GAN performance. We then use automated Question Generation and Answering on these summaries and the ground truth summary to evaluate their goodness. We use the CNN/Daily Mail dataset for our summarization task.

1. Introduction

TL; DR is a frequent issue for all of us in this information age. With the advent of technology and the internet, we have moved from a time when information was rare and inaccessible to the other end where we have too much information. As always too much information is not a bad thing unless you don't process it. In order to better manage our time and effectively process all the information that is out there, we need effective summarization techniques for text documents. Text summarization has its applications in search engines, news headlines, scientific paper abstracts, conversational summaries etc. There are two approaches to automatic text summarization, extractive and abstractive. Extensive and state of the art research has been carried out for extractive summarization while abstractive summarization models are still evolving. The focus of our project is to compare the performance of two abstractive automatic text summarization models using a cognitive semantic approach in addition to traditional quantitative metrics.

We use two neural network models for our analysis. One is a Seq2Seq Pointer Generator RNN. Seq2Seq is traditional Machine Translation Model [2], [3], [6] that converts a variable length input to a variable length output. We incorporate the same model with sentence level attention with coverage for the automatic text summarization task.

Our second network is the state-of-the-art Generative Adversarial Network that trains two neural networks to become better at a given task. In our version of GAN[1] for automatic text summaries we will use a bi-directional LSTM with coverage attention mechanism as a Generator. A binary classifier with input features built using a LSTM is the discriminator that tries to differentiate the Generator Summaries from our human generated ground truth summaries. The Generator's goal is to try to fool the discriminator classifier and this competitive training achieves good results on the task.

Once we have generated summaries using our two models, we will use a text generation utility to generate questions on our ground truth summary. We will then have humans use our model summaries to answer the questions generated by us. In comparing the answers and the overall performance of both the systems we can identify which network performs better on this task much better than the quantitative ROUGE scores which scores on intersecting N-grams that do not capture the semantic meaning of the summaries [4].

2. Background/Related Work

There is plenty of research going on in extractive summarization [8]. For our Seq2Seq RNN we draw inspiration from the Pointer Generator Model developed in [6]. The work discusses using LSTM Encoder-Decoder architecture to perform abstractive text summarization. Two issues identified by this work are the problems in encountering out-of-vocabulary words and the tendency of the summaries to repeat themselves. In resolving the out-of-vocabulary words, the paper uses a predefined list of vocabulary words and extends them for each article from the dataset using the words in each of the articles respectively. This is different from other works which did not use words from the original news story article. In order to prevent the summaries from generating duplicate tokens, the work introduces Coverage Attention mechanism where it uses attention from previous steps to compute the attention for the current timestep.

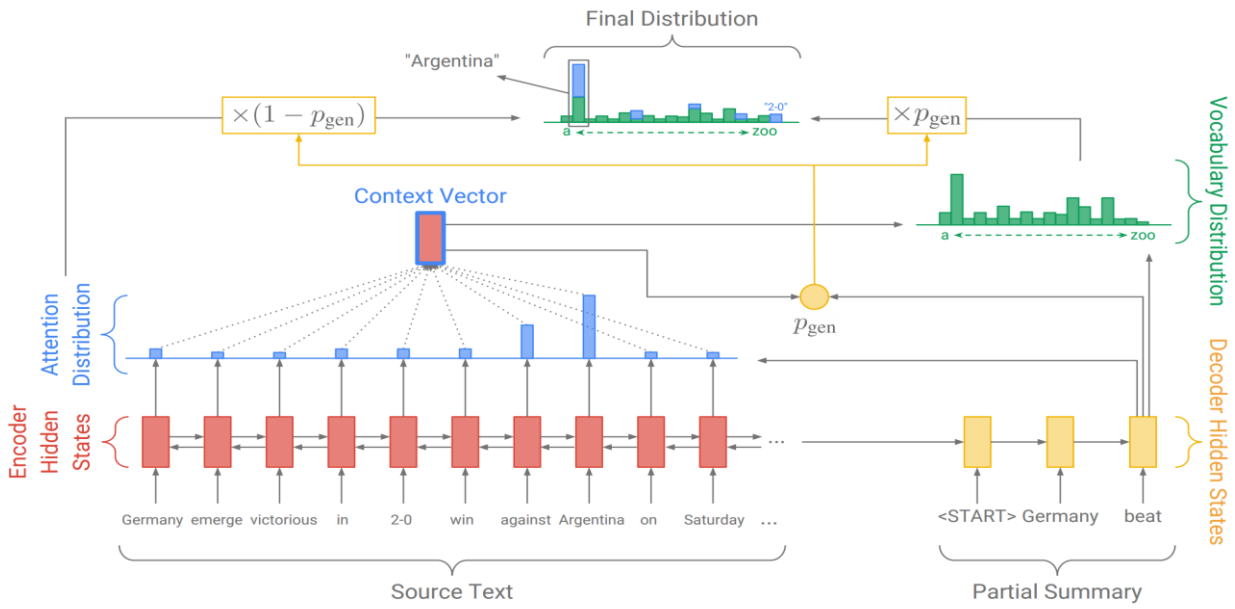

Figure 1 from [6]: Pointer-generator network: For each decoder timestep a generation probability pgen ∈ [0,1] is calculated, which weights the probability of generating words from the vocabulary, versus copying words from the source text. The vocabulary distribution and the attention distribution are weighted and summed to obtain the final distribution, from which we make our prediction

For our GAN, we follow the minimax strategy used in traditional GAN's where the generator and the discriminator are trained in an alternating manner. One of the common issues in training LSTM's in an adversarial manner is that the input to the LSTM is in a discrete space and making small changes to this is not possible. Hence to overcome this non-differentiable issue, the paper in [1] uses Reinforcement learning to update the parameters of G. They use a policy gradient that we adapt for our models as well. The paper trains the Generator and Discriminator alternatively to make small adjustment in the Generator that can help in building effective abstractive summaries.

Our work incorporates the work from the Pointer Generator RNN model with changes in the parameter update function and the attention mechanism that we used. The original work uses AdaGrad for per-parameter update. But because of the nature of AdaGrad to stop learning we use the well experimented Adam update rule. Our work on GAN uses an LSTM based Text Classifier instead of the CNN-Text classifier for the discriminator. This change is so that the model learns long-term dependencies in summaries as abstractive summaries tend to have long sentences.

The most common evaluation metric for abstractive text summarization has been ROUGE (Recall-Oriented Understudy for Gisting Evaluation). This metric uses words, word-bigrams or word sequences that are in common between the generated summary and the ground-truth summary to define a numerical score. The interpretability on the score does not evaluate the semantic underpinnings of the text. The score also fails to penalize word repetitions.

For our semantic evaluation technique, we incorporate the idea proposed by [4]. The key idea behind such an evaluation is that using QA to identify differences gives us a sense of the content that differs in the summaries. They generate questions on the original source text and use the generated summaries as the knowledge source to the QA system. They use an existing OpenEphyra framework for generating answers.

Our work does not use the original source text to generate the questions instead we use the ground truth summaries. The question generation tool uses named entities to generate the questions, using the entire original news story from our CNN/Daily Mail dataset as the source will make our ground truth summaries also fail in answering questions that were not present in them. So, to avoid this discrepancy we use the ground truth summary to generate the questions so that in evaluation the ground truth summary will have the perfect score for answering, as the questions were originally from them.

3. Approach

In this section we describe our approach to the following: (1) Data Preprocessing, (2) Pointer Generator Seq2Seq RNN, (3) Our LSTM based GAN, (4) our semantic evaluation technique.

3.1. Data Pre-processing

The primary sources of data for us are the CNN/DailyMail dataset [14]. The stories in the dataset need to be tokenized, lowercased and written to serialized binary files (train.bin, val.bin and test.bin) before starting to train our models on the data. Additionally, we require a vocab file which is a dictionary of all the unique words and their frequency. After generating the binary files, they are split into chunks of 1000 examples per chunk.

For our baseline summary (Lead30-BL) we extract the top 30% of our original news story from the CNN/Daily Mail dataset. For our Pointer Generator Seq2Seq RNN we use a predefined list of 50K words as our vocabulary list.

Although the CNN/Daily Mail dataset contains questions for their stories, these questions are place-holder based questions and may not help us semantically understand our summaries. To this effect we use our own Question Generation and Answering system for evaluation.

3.2. Pointer Generator Seq2Seq RNN

Our pointer-generator model uses the same architecture as our reference from [6],[12] as depicted in Figure 1. The tokens of the article w_i are fed one-by-one into the encoder, producing a sequence of encoder hidden states h_i. At each time step t, the decoder receives the word embedding of the previous word (while training, this is the previous word of the reference summary; at test time it is the previous word emitted by the decoder) and has decoder state s_t. Then, we calculate the Bahdanau attention distribution a^t as in [3]:

$$e_i^t = v^T \tanh(\mathrm{W_h} h_i + W_s s_t + b_{attn}) \quad (1)$$

$$a^t = softmax(e^t) \quad (2)$$

The attention distribution is a probability distribution over the source words that tells the decoder where to focus its efforts in – in the original text. Using the attention distribution as weights we compute a context vector h_t^* which is the linear combination of the attention weights with the encoder hidden states:

$$h_t^* = \Sigma_i a_i^t h_i \quad (3)$$

Then, the context vector is concatenated with the decoder state s_t and fed through two linear layers to produce the final probability distribution P_{vocab} from which we predict words:

$$P_{vocab} = \mathrm{softmax}(V'(V[s_t, h_t^*] + b) + b') \quad (4)$$

where V, V', b and b' are learnable parameters. During training, we use the negative log likelihood loss for each target word w_t^* at timestep *t:*

$$loss_t = -\log P(w_t^*) \quad (5)$$

In addition, we calculate a *generation probability* $p_{gen} \in [0,1]$ for each timestep t using the context vector h_t^*, the decoder state s_t and the decoder input x_t:

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}) \quad (6)$$

where vectors w_h^*, w_s, w_x and scalar b_{ptr} are learnable parameters and σ is the sigmoid function. We can think of p_{gen} as a soft switch used to choose between *generating* a word from the vocabulary by sampling from P_{vocab}, or *copying* a word from the input sequence by sampling from the attention distribution a^t. For each document let the *extended vocabulary* denote the union of the vocabulary, and all words appearing in the source document. We obtain the following probability distribution over the extended vocabulary:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \quad (7)$$

Note that in this case, if w is an out-of-vocabulary (OOv) word, then $P_{vocab}(w)$ is zero; similarly, if w does not appear in the entire source document, then the second term becomes zero. The ability to produce OOV words is one of the main advantages of the pointer generator model.

This pointer generator model helps us generate a varied selection of words. But one of the most common issues in Seq2Seq RNN models are the repetition of words in the summary. Our reference paper from [6] uses a coverage-based attention mechanism to combat this issue. *Coverage vector* c^t is the sum of all the attentions computed in the previous timesteps for the decoder. This computed vector is used as an extra input to the attention mechanism, changing equation (1) to:

$$e_i^t = v^T \tanh(\mathrm{W_h} h_i + W_s s_t + w_c c_i^t + b_{attn}) \quad (8)$$

where w_c is a learnable parameter vector of same length as v. In addition, we define a coverage loss which is set to penalize those attention vectors that are similar in nature, i.e if the attention is repeated the coverage loss increases, this prevents from recomputing the same nature of attention or using the same words repeatedly in the summary. This coverage loss forces the network to learn from new areas in the original text. Hence the new composite loss function

from (5) and including the coverage loss and the negative log likelihood is given as:

$$loss_t = -\log P(w_t^*) + \lambda \sum_i \min(a_i^t, c_i^t) \quad (9)$$

Also, during the decoder step at test time, we use the technique of beam search decoding. Instead of greedily choosing the most likely next step as the sequence is constructed, the beam search expands all possible next steps and keeps the k most likely, where k is a user-specified parameter and controls the number of beams or parallel searches through the sequence of probabilities.

3.3. TextGAN

For our text-based GAN built using [11], [12] we simultaneously train two models in an adversarial manner, a generative model G and a discriminative model D.

Generator

For the generator network, we build a bi-directional Seq2Seq LSTM Encoder and Decoder architecture with Bahdanau Attention with Coverage similar to our Seq2Seq RNN, however we do not use a pointer-generator mode in this case. The use of the discriminator and the adversarial training we believe will eradicate the repetition of text in the generated summaries instead. Hence, we define the final loss for the Generator same as our Seq2Seq RNN, but with coverage

$$loss_t = -\log P(w_t^*) + \lambda \sum_i \min(a_i^t, c_i^t) \quad (10)$$

Discriminator

For our discriminator we use a unidirectional-single layer LSTM. This is different from the reference paper in [1]. We replaced the CNN classifier with an LSTM classifier so that the summaries which are long sentences are better represented. This LSTM extracts features. It is then put through a SoftMax layer to get the distribution across two-classes. The input to the Discriminator is the summaries generated with G and the ground-truth summaries.

Adversarial Training

In order to overcome the differential difficulties with LSTM GANs we use the same method as used in [1] for updating the model parameters using reinforcement learning. We minimize D as:

$$\min_{\emptyset} -E_{Y\sim p_{data}}[log D_{\emptyset}(Y)] - E_{Y\sim G_{\emptyset}}[log(1 - D_{\emptyset}(Y)] \quad (11)$$

After the update for D, the loss for G includes both loss from the policy gradient and the maximum likelihood loss (Eq. 12).

$$J = \beta J_{pg} + (1 - \beta) J_{ml} \quad (12)$$

The reward for G, a stochastic parametrized policy, is the score given by D. Using the REINFORCE algorithm, the paper takes the estimated probability of the summary generated being the ground truth summary as given by D as the reward function:

$$R_D^{G_\theta}\big((Y_{1:t-1}, X), y_t\big) = D(X, Y_{1:t}) - b(X, Y_{1:t}). \quad (13)$$

b (x, y) in (Eq. 13) denotes the baseline value to reduce the variance of the reward while keeping it unbiased. So, for optimizing the generator, it wishes to maximize its reward:

$$J_{pg}(\theta) = \sum_{t=1}^{T} G_\theta\left(y_t|(Y_{1:t-1}, X)\right) R_D^{G_\theta}\big((Y_{1:t-1}, X), y_t\big) \quad (14)$$

In addition to the policy gradient loss we use the Seq2Seq loss of cross entropy or negative log likelihood.

$$J_{ml}(\theta) = -\sum_{t=1}^{T} logp(y_t^*|y_1^*, y_2^*, \dots, y_{t-1}^*, x). \quad (15)$$

Similar to our Seq2Seq RNN at inference for our Generator we use beam search decoding for test time sampling. This almost identical architecture helps us make a fair comparison between the two neural network models.

3.4. Semantic Evaluation

In order to overcome the limitations of ROUGE [13] based evaluation, our semantic evaluation makes use of NLP techniques such as Question Generation and Question Answering.

We incorporate an automatic Question Generation System. Question Generation using the semantics of the text is in itself a core-NLP problem. In the interest of time we adapt an existing Question Generation component [5]. We use our CNN/Daily Mail dataset after data preprocessing to generate questions on the ground truth summaries. We generate only 'Wh' questions on the ground truth summary and then measure how many of these questions are answered by the summaries generated using our two models.

4. Experiments

In this section we provide the experimental details for our models (1) Pointer-Generator RNN, (2) TextGAN, (3) Semantic Evaluation. Our CNN/Daily Mail dataset contains 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs. We restrict all our generated summaries

to be less than 100 tokens. We used the Google Compute Instance to train all our models with one NVIDIA Tesla P100 GPU.

4.1. Pointer-Generator RNN

Our pointer generator model has 256-dimensional hidden states and 128-dimensional word embeddings. We used a vocabulary of 50k words for both our source and target. Note that the coverage mechanism does not add many extra parameters. The baseline model has 21,499,600 parameters, the pointer-generator adds 1153 extra parameters (w_h, w_s, w_x, and b_{ptr} in equation 6), and coverage adds 512 extra parameters (w_c in equation 8). We trained the model using the Adam optimizer with learning rate 0.001, beta1 = 0.9 and beta2=0.999 (default from TensorFlow) with a batch size of 25. We trained the baseline model for about 380,000 iterations (20 epochs), which took around 4 days and 7 hours. We added the coverage mechanism with $\lambda = 1$(Eq. 9) after initial training and trained for a further 10,000 iterations (around 9 hours). During training and test time, we set the max_enc_steps to 400 i.e. truncating the article to 400 tokens and max_dec_steps (tokens in summary) to 100. After training, we used beam search decoding at test time to generate summaries with beam size 4. Below is a summary generated by our model compared to the reference:

Generated Summary*: new: a suicide attacker detonated a car bomb near bolan city in september. the attack happened at about 6 p.m. in the bolan area of lashkar gah city .the attack happened in a farming community in bolan city .*

Reference Summary: *the attack happened at about 6 p.m. in the bolan area of lashkar gah city. several children were among the wounded , and the majority of casualties were civilians , police said .*

After generating our test summaries, we calculated the ROUGE-1, ROUGE-2 and ROUGE-L scores to evaluate our RNN generated summaries and our ground-truth summaries.

	ROUGE-1	ROUGE-2	ROUGE-L
Lead30-BL	39.48	17.24	35.68
Pointer-generator+coverage	**37.32**	**16.77**	**31.91**
TextGAN+coverage	30.12	10.64	27.43

Table 1: Comparing ROUGE scores of the baseline summaries, summaries generated by the pointer-generator with coverage, and the summaries generated by our TextGAN.

4.2. TextGAN

To implement the Generator G for our TextGAN we follow a setup similar to that of RNN. We run our experiments with two types of word embeddings, (1) GLOVE pre-trained word embedding [9], (2) Learning them through the entire training process.

For our pre-trained GLOVE word-embedding we used word-embedding of 300 dimensions. We used the pre-trained model on Wikipedia 2014 + Gigaword 5 model with 400K vocabulary words. In comparison to the original reference paper that uses 128 dimensions for word embeddings for learning them from scratch, we increased it to 300 to compare the performance with GLOVE embeddings. The vocabulary size for the learned word-embeddings are restricted to the words in the articles.

After running the entire GAN for 2 days on the Google Compute instance we found that the training time for learning the word-embeddings from scratch was much longer than using the GLOVE embeddings. So, we used the GLOVE embeddings to train our GAN with coverage attention $\lambda = 1$(Eq. 9) for the Generator and LSTM for classifier for about 320,000 iterations (17 epochs) on the Google Compute instance.

We used hidden-units of 256 for all our LSTM models. With Adam optimizer with learning rate 0.001, beta1 = 0.9 and beta2=0.999 (default from TensorFlow) with a batch size of 25. During training and test time, we set the max_enc_steps to 400 i.e. truncating the article to 400 tokens and max_dec_steps (tokens in summary) to 100. After training, we used beam search decoding at test time to generate summaries with beam size 4.

A sample summary generated by our GAN:

Generated Summary*: the attack happened at about 6 p.m. in the bolan area of lashkar gah city. several children were among the wounded , and the majority of casualties were civilians , police said .the attack happened at about 6 p.m. in the bolan area of lashkar .*

Reference Summary: *the attack happened at about 6 p.m. in the bolan area of lashkar gah city. several children were among the wounded , and the majority of casualties were civilians , police said*

As we can observe, the summaries still have repetitions of words and even whole sentences.

We then calculated ROUGE-1, ROUGE-2 and ROUGE-L scores to evaluate our GAN generated summaries and our ground-truth summaries.

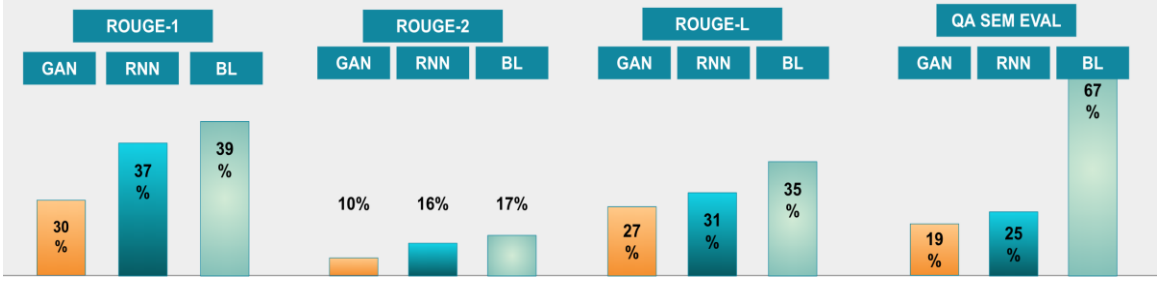

Figure 2: Comparing the scores from using ROUGE and semantic QA evaluation on our 3 sets of generated summaries: the Lead30 baseline(BL) summary, summary generated by TextGAN and summary generated by our pointer generator RNN

4.3. Evaluation using ROUGE scores and QA-QG

We use ROUGE-1, ROUGE-2 and ROUGE-L scores to evaluate our models quantitatively and serve as baseline metrics. The scores for both of our models along with Lead30-BL summary which we generated by extracting the top 30% of the source document is tabulated in Table 1. The ROUGE scores indicate that our pointer generator model performs on a level close to the Lead30-BL and performs much better than our TextGAN in generating summaries.

For our semantic evaluation, we calculate the percentage of questions answered by the summaries from the two models to compare their performance using manual identification of whether the summaries can generate answers for the questions.

We take 50 samples from our ground truth summaries and generate questions on them using our modified version of Heilman's Question Generation (QG) tool [5]. Some of the questions generated using the QG tool on the ground truth summaries were bad. In order to effectively evaluate the generated summaries, we restricted our QG output to 5 questions per ground truth summary. A sample output from the QG tools is listed below along with some findings.

For an article on the Ghadafi Family; We generated 20 'WH' questions from the ground truth summary. Some of the questions generated were:

Good Questions:

1. *Who were able to flee Libya into Algeria?*
2. *Who was Gadhafi's wife?*
3. *What did U. N. Security Council resolution forbid to the Gadhafi family?*

Bad Questions:

1. *Whose wife was three children?*
2. *Whose wife was three children? (repeated)*
3. *What forbade travel to the Gadhafi family?*

Observations:

Most of the summaries generate around 20-25 questions. Of which 3-5 are duplicate and 5-7 are bad questions.

Using the questions generated we evaluate three generated summaries: 1. Lead30BL, 2. Pointer-Generator RNN, 3. TextGAN

We use 50 generated summaries from each model, and each summary has 5 associated questions. So, we get a score out of 250 for both the models based on how many questions are answered by the summary. If a model generated summary contains the answer, we increment the model score by 1. The percentage of correctly answered questions i.e. $modelscore/250$ gives the QA-score for the model. The semantic evaluation scores along with the ROUGE scores for the three sets of summaries are presented in Figure 2.

We can see from the figure that while ROUGE scores indicate that our model generated summaries, especially by the pointer-generator, are almost as good as the simple extractive Lead30 BL summary, the semantic evaluation scores using question answering paints a different picture. Our models performed significantly worse at question answering compared to the extractive baseline summary. While doing the semantic evaluation, we identified a couple of reasons why this may be the case. First, ROUGE scores do not penalize any repetitions of words/phrases in the summary and in fact the scores get higher if there are repetitions, due to the way ROUGE scores are calculated. Second, ROUGE scores do not factor in the semantic meaning and cohesiveness of the summaries, instead purely relying on comparing word-to-word. When we did the semantic evaluation, the scores for our model generated summaries went down because while the words in the summary were same as the reference, the words were jumbled and repeated on occasion, and some sentences did not make any sense, resulting in fewer answers to our questions and thus getting a low score.

This shows the ineffectiveness of using only ROUGE scores to measure the goodness of generated summaries.

5. Conclusion

In our experiments the Pointer-Generator model did better quantitatively and qualitatively compared to GAN. This is because of the repetitive sentences generated in GAN with its simple architecture. Additional features like

pointer-generator in our RNN could have been the factor in getting superior results over GAN. Between the models, even though a GAN theoretically should provide good results, it can be seen that the power of the GAN comes from having a strong generator as well.

Although our RNN produced good ROUGE-* scores, when compared semantically with the Lead30 Baseline their capability to answer questions was very low. The reason for very high ROUGE-1 and ROUGE-L scores are that the metric only focusses on the common words found between the generated and the reference summary and not on the semantic meaning of the summary. So even though our abstractive techniques generated new words from the article representations they failed to perform semantically better than our baselines. One of the reasons our naïve baseline seems to capture effective information is the fact that the dataset contains news stories, and they are generally written in a way to contain as much information about the content in the initial few sentences to capture the reader's attention. Our naïve baseline on another dataset might not be as effective.

This shows that the ROUGE is not an effective metric for abstractive text summarization. And the models used for abstractive summarization are capturing way less information than a naïve extractive baseline summary.

Future Work

1. RNN – Use pre-trained word-embeddings like GLOVE or FastText – this has better word representation than the ones we are learning.
2. GAN – Mock up the architecture of the Pointer-Generator RNN in addition to the LSTM classifier, i.e. use Pointer Generator
3. GAN – Implement FastText Word Embeddings, these are sub-gram embeddings and perform much better on out-of-vocabulary words.
4. Use a Text Comprehension Neural Network such as DocumentQA [10] to perform the QA task in semantic evaluation.

6. References

[1] [Liu et al. 2018] Liu L; Lu Y; Yang M; Qu Q; Zhu J; Li H. Generative Adversarial Network for Abstractive Text Summarization. arXiv preprint arXiv: 1711.09357v1

[2] [Nallapati et al. 2016] Nallapati, R.; Zhou, B.; Gulcehre, C.; Xiang, B.; et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023.

[3] [Bahdanau et al. 2014] Dzmitry Bahdanau; Kyunghyun Cho; and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473

[4] [Chen P et al. 2018] Chen P; Wu F; Wang T; Ding W. A semantic QA-based Approach for Text Summarization Evaluation. arXiv preprint arXiv:1704.06259

[5] [M. Heilman et al. 2011] M. Heilman. 2011. Automatic Factual Question Generation from Text. Ph.D. Dissertation, Carnegie Mellon University. CMU-LTI-11-004. http://www.cs.cmu.edu/~ark/mheilman/questions/

[6] [See A et al. 2017] Abigail See; Peter J. Liu; Christopher D. Manning. 2017. Get to The Point: Summarization with Pointer-Generator Networks. arXiv preprint arXiv: 1704.04368v2

[7] Tensor Flow Seq2Seq Model for Summarization: https://github.com/tensorflow/models/tree/master/research/textsum

[8] [Verma et al. 2017] Sukriti Verma; Vagisha Nidhi; Extractive Summarization using Deep Learning. 2017. arXiv preprint arXiv:1708.04439

[9] [Pennington et al. 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

[10] [Clark et al. 2017] Chistopher Clark, Matt Gardner. Simple and Effective Multi-Paragraph Reading Comprehension. arXiv preprint arXiv:1710.10723

[11] https://github.com/amirbar/rnn.wgan/blob/master/model.py

[12] https://github.com/google/seq2seq

[13] https://github.com/bheinzerling/pyrouge

[14] https://github.com/abisee/cnn-dailymail

This project is not shared with any other class at UMASS