Abstractive Text Summarization

Soumye Singhal
Department of Computer Science
IIT Kanpur
soumye@cse.iitk.ac.in

Arnab Bhattacharya
Department of Computer Science
IIT Kanpur
arnabb@cse.iitk.ac.in

Abstract

Neural Sequence to Sequence attention models have shown promising results in Abstractive Text Summarization. But they are plagued by various problems. The summaries are often repetitive and absurd. We explore and review different techniques that can help overcome these issues. We also explore a Reinforcement Learning based Training procedure using intra-Attention that significantly improves the model's performance. We analyze the problems that plague the area in detail, reasons and possible ways to improve upon. We also propose a novel architecture to solve the problem of long summary generation, uncaptured by the current models.

1 Introduction

In the modern Internet age, textual data is ever increasing. Need some way to condense this data while preserving the information and meaning.We need to summarize textual data for that. Text summarization is the process of automatically generating natural language summaries from an input document while retaining the important points. It would help in easy and fast retrieval of information.

There are two prominent types of summarization algorithms.

- **Extractive summarization** systems form summaries by copying parts of the source text through some measure of importance and then combine those part/sentences together to render a summary. Importance of sentence is based on linguistic and statistical features.
- **Abstractive summarization** systems generate new phrases, possibly rephrasing or using words that were not in the original text. Naturally abstractive approaches are harder. For perfect abstractive summary, the model has to first truly understand the document and then try to express that understanding in short possibly using new words and phrases. Much harder than extractive. Has complex capabilities like generalization, paraphrasing and incorporating real-world knowledge.

Majority of the work has traditionally focussed on extractive approaches due to the easy of defining hard-coded rules to select important sentences than generate new ones. Also, it promises grammatically correct and coherent summary. But they often don't summarize long and complex texts well as they are very restrictive.

2 Overview of Deep Learning Approaches

The traditional rule-based AI did poorly on Abstractive Text Summarization. But the recent advances in Deep Learning changed that for the good. Inspired by the performance of Neural Attention Model in the closely related task of Machine Translation Rush et al. [2015] applied this Neural Attention Model to Abstractive Text Summarization and found that it already performed

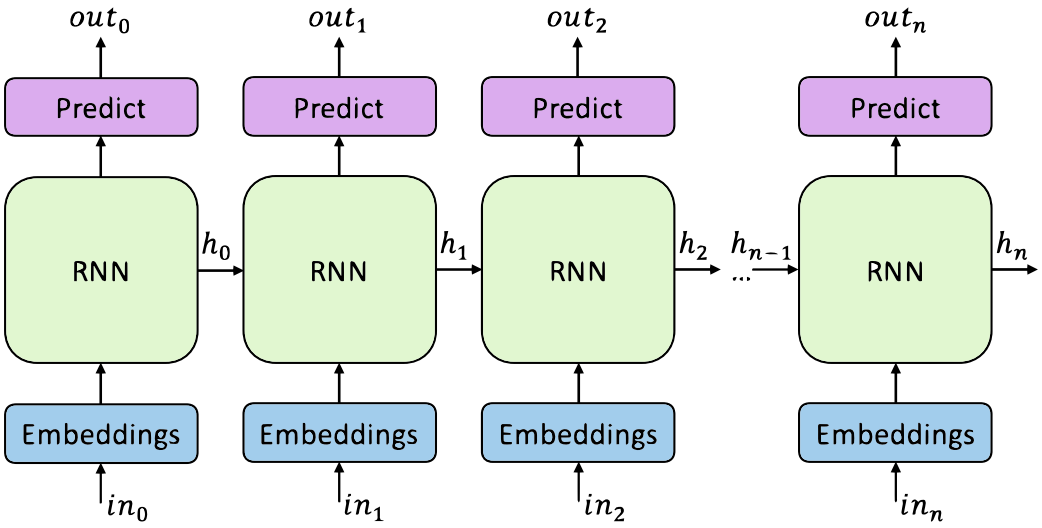
very well and beat the previous non Deep Learning-based approaches. But they applied this only to single sentences and didn't generalize. Then Nallapati et al. [2016] generalized this further and set the baseline Model. We explore this model in detail and then see it's shortcomings. Further improvements have been done to this baseline model by See et al. [2017] using Pointer Generator Networks and Coverage Mechanism and by Paulus et al. [2017] using Reinforcement Learning based Training procedure and intra-Attention on decoder outputs as well. Both these approaches significantly improve the ROUGE scores over the baseline.

3 Baseline Neural Attention Model

The Neural Attention Model as introduced by Bahdanau et al. [2014] consists of an encoder-decoder RNN with attention Mechanism. The inputs that we feed into the RNN are word/morpheme/phrases embeddings. We use word embeddings for summarization.

3.1 Simple Encoder-Decoder Model

Figure 1: A simple Recurrent Neural Network Unit unrolled

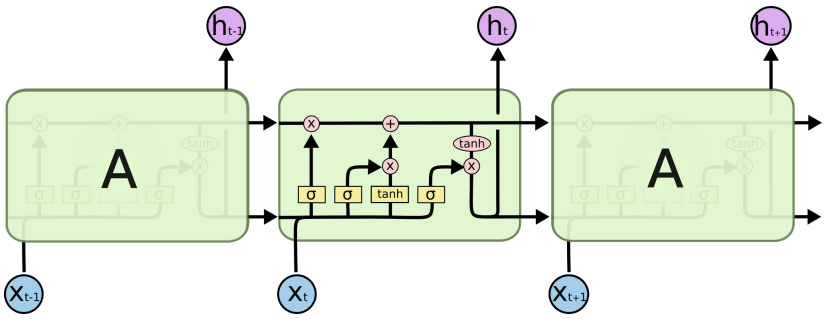


- w_i - input tokens of source article
- h_i - Encoder hidden states
- $P_{vocab} = softmax(Vh_i + b)$ is the distribution over vocabulary from which we sample out_i

There are many variants of RNN's like LSTM's and GRU's and NAS. A detailed view of an LSTM unit with all of its layers is described below. The motivation behind using an LSTM is that it captures long-term dependency pretty well and the information in the starting of the sequence is able to traverse down the line. This is done by selectively selecting and restricting the flow of information in the LSTM unit. There are three gates in an LSTM.
Forget Gate Layer

- $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
- $C_t = C_t \otimes f_t$

Figure 2: A Basic LSTM Unit with it layers.



Input Gate Layer [1]

- $i_t = \sigma(W_t[h_{t-1}, x_t] + b_i)$
- $\widetilde{C_t} = tanh(W_i[h_{t-1}, x_t] + b_c)$
- $C_t = \widetilde{C_t} \otimes t_t + C_t$

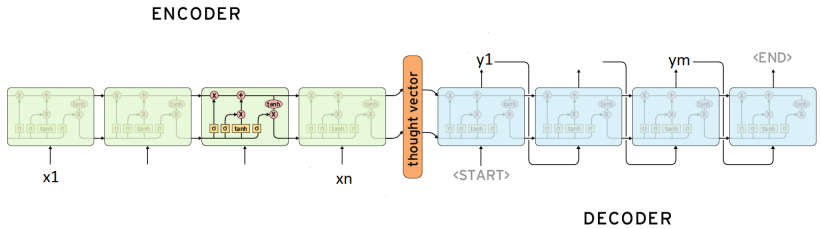
Output Gate Layer

- $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
- $h_t = o_t * tanh(C_t)$

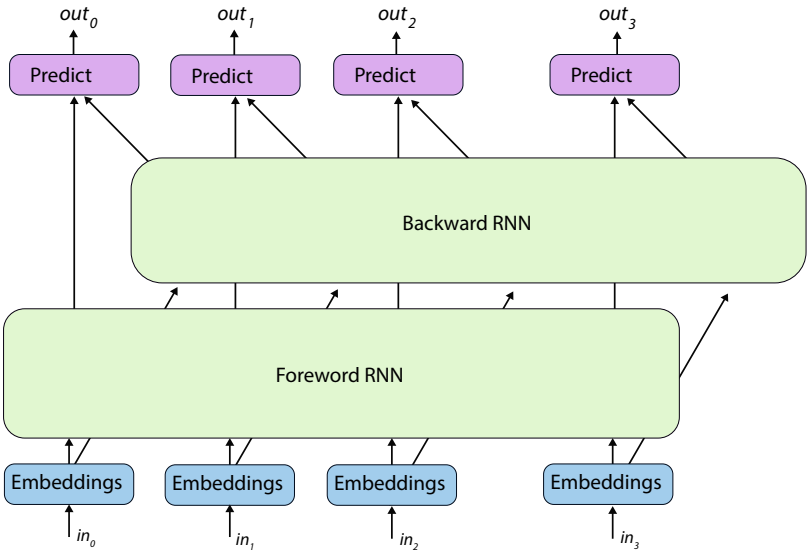
We also use **Bidirectional** RNN's which scan the input from both left and right. This done so that at any step both the words to the right and words to the left are included in the context.

- We make two passes on the source sequence, computing hidden states for backward pass $\overleftarrow{h_t}$ and for forward pass $\overrightarrow{h_t}$
- We then concatenate both of them to get final encoder hidden state. $h_t = [\overleftarrow{h_t}, \overrightarrow{h_t}]$ now encodes both past and future information.

In the vanilla **encoder-decoder model**, the encoder RNN first reads in the source string word by word, encoding the information in a hidden state and passing the context forward. On a complete pass, the encoder ie the Bi-Directional RNN produces an encoding of the string or a though vector so to speak which captures all the information and context of the input string. The decoder is another RNN which learns to decode the thought vector into output sequence.



[1] Image taken from `colah.github.io`



3.2 Attention Mechanism

The basic encoder-decoder model performs okay on very short sentences but it fails to scale up.

- The main bottleneck is the fixed sized encoding of the input string, which is the LSTM output of the last time-step. Due to it's fixed size it is not able to capture all the relevant information of the input sequence as the model sizes up.
- At each generation step, only a part of the input is relevant. So how does the model decide which part of the input encoding to focus on at each generation step?
- At each step, the decoder outputs hidden state h_i, from which we generate the output.

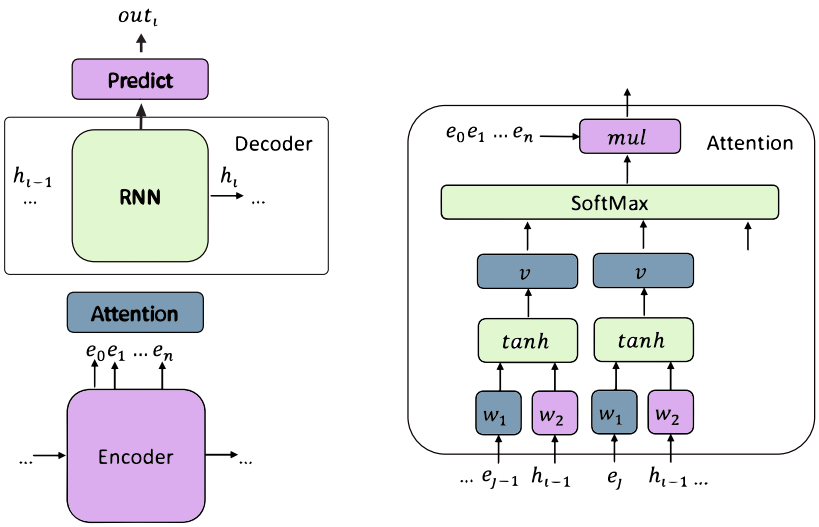
The solution to this problem is using the attention model. This model calculates the importance of each input encoding for the current step by doing a similarity check between decoder output at this step and input encodings. Doing this for all of the input encodings and normalizing, we get an importance Vector. We then convert it to probabilities by passing through softmax. Then we form a context vector by multiplying with the encodings.

- $importance_{it} = V * tanh(e_i W_1 + h_t W_2 + b_{attn})$.
- Attention Distribution $a^t = softmax(importance_{it})$
- ContextVector $h_t^* = \sum_i e_i * a_i^t$

[2]. Context Vector is then fed into two layers to generate distribution over the vocabulary from which we sample.

- $P_{vocab}(w) = softmax(V'(V[h_t, h_t^*] + b) + b')$
- For the loss at time step t, $loss_t = -\log P(w_t^*)$, where w_t^* is the target summary word.
- $LOSS = \frac{1}{T} \sum_{t=0}^{T} loss_t$ is the total loss across all timesteps.
- We then use the Backpropagation through time Algorithm to get the gradients and then apply any of the popular Gradient Descent algorithms to minimize the loss and learn good parameters.

[2]Image stylized from https://talbaumel.github.io/attention/



Generation of Summaries
At each step, the decoder outputs a probability distribution over the target vocabulary. To get the output word at this step we can do the following

- Greedy Sampling, ie choosing the mode of the Distribution
- Sampling from the distribution.
- **Beam Search** - Choosing the top k most likely target words and then feeding them all into the next decoder input. So at each time-step t the decoder gets k different possible inputs. It then computes the top k most likely target words for each of these different inputs. Among these, it keeps only the top-k out of k^2 and rejects the rest. This process continues. This ensures that each target word gets a fair shot at generating the summary.

4 Metrics

We need to evaluate the quality and coherence of the summary. By quality, we mean how well does the summary capture the source document and by coherence we mean if it's grammatically correct and is human readable. In the dataset, for a given document we may or may not be given a golden *target summary*.

- If the *target summary* is not given, then we need to make a similarity measure between the summary and the source document. The crux is that in a good summary, the topics covered would be roughly the same as that of the topics covered in the document. So we may use topic models like Latent Semantic Analysis(LSA) and Latent Dirichlet Allocation(LDA) to measure the similarity between topics.
- If the *target summary* is given then it is better to use metrics like `ROUGE` and `METEOR`. They are essentially string matching metrics. `ROUGE` is the most popular and has various variants like `ROUGE-N` and `ROUGE-L`. `ROUGE-N` measures the overlap of N-grams between the system and reference summary. A bigger `N` implies more fluency in the summary, because matching with a bigger portion of the reference summary which is more fluent, implies more fluency. `ROUGE-L` is based on longest common subsequences, thereby also taking into account sentence level similarity. `ROUGE-S` is the skip-gram variant which though not very popular currently but does seem promising to use for longer summaries.

5 Datasets

There are two popular Sentence level Datasets for summarization.

- DUC-2004
- Gigaword

Nallapati et al. [2016] adapted the CNN/Daily Mail Dataset for summarization task, which is most commonly used today. There is also a less popular NYT dataset.

6 Problems

Though the baseline gives decent results, they are clearly plagued by many problems

1. They sometimes tend to reproduce factually incorrect details.
2. They struggle with Out of Vocabulary (OOV) words. So we see many `UNK` tokens in the summary.
3. They are also a bit repetitive and focus on a word/phrase multiple times.
4. Focus is mainly on single sentence summary tasks like headline generation.

7 Improvements

Large Vocabulary Trick and Feature-rich Encoder

Introduced by Nallapati et al. [2016] In order to capture more meaning into the inputs fed into the encoder, we go beyond the word-embeddings based representation of the input like word2vec or GloVe and also incorporate more linguistic features like POS(parts of speech) tags, named-entity tags, and TF-IDF statistics. Though it speeds up training, it hurts the abstractive capabilities of the model. When these tags are given the model is able to converge faster. But it's surprising as to why the abstractive capabilities decrease.

Hierarchical Attention

It was first used Nallapati et al. [2016]. Based on the idea that for the summary some sentences are more important than others. So they use two Bi-Direction RNN to scan the source text, one at word level and another at the sentence level. Then the calculate word-level attention using first encoder and sentence level attention using the second encoder. Word level attention is then weighted by corresponding sentence level attention.

$$P^a(j) = \frac{P^a_w(j)P^a_s(s(j))}{\sum_{k=1}^{N_d} P^a_w(k)P^a_s(s(k))}$$

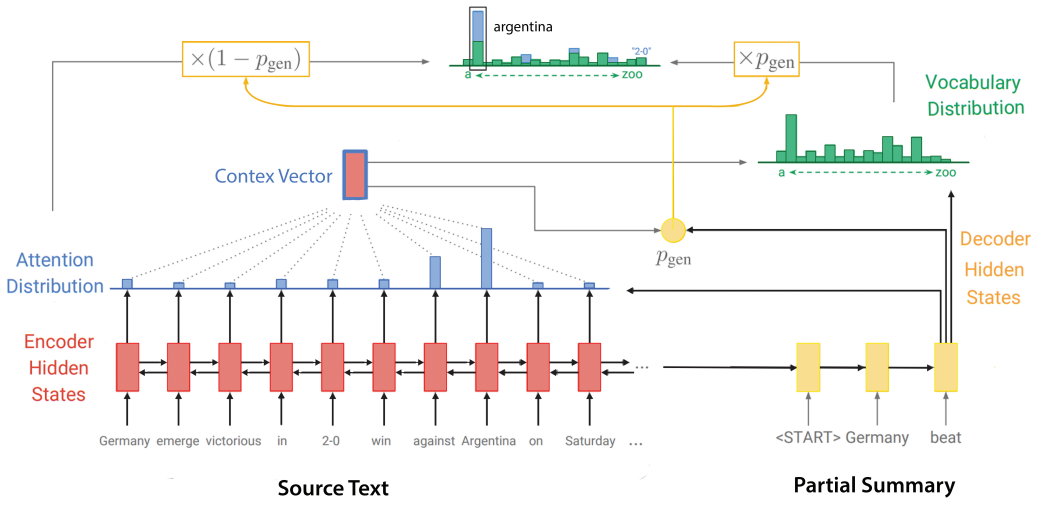
Here $P^a(j)$ is the attention given to word j where it's corresponding sentence is $s(j)$. P^a_w is the word level attention and P^a_s is the sentence level attention.

Pointer Generator Network

Introduced by Nallapati et al. [2016] and See et al. [2017]. It helps to solve the challenge of OOV words and factual errors.It works better for multi-sentence summaries.The basic idea is to choose between generating a word from the fixed vocabulary or copying one from the source document at each step of the generation. It brings in the power of extractive methods by pointing (Vinyals et al. [2015]). So for OOV words, simple generation would result in UNK, but here the network will copy the OOV from source text. [3] At each step we calculate generation probability p_{gen}

$$p_{gen} = \sigma(w^T_{h*}h^*_t + w^T_s h_t + w^T_x x_t + b_{ptr})$$

[3] Image taken from blog, `www.abigailsee.com`



Here x_t is the decoder input. The parameter $w_{h^*}, w_s, w_x, b_{ptr}$ are learnable. Now this p_{gen} is used as a switch. We get the final distribution for output word as follows.

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

Note that for OOV word $P_{vocab}(w) = 0$, so we end up pointing.

Coverage Mechanism

This concept was first applied by See et al. [2017]. The cause of repetitiveness of the model can be accounted for by increased and continuous attention to a particular word. So we can use `Coverage Model` by Tu et al. [2016]. We form a Coverage vector as follows,

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

Intuitively, by summing the attention at all steps we are keeping track of how much coverage each encoding, e^i has received. Now, give this as input to attention mechanism. The updated formulate for calculating importance is,

$$importance_{it} = V * tanh(e_i W_1 + h_t W_2 + W_c c_i^t + b_{attn})$$

Now to make the network learn to not focus on things that have already been covered we penalize attending to things that have already been covered.

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

This loss penalizes overlap between attention at this step and coverage till now. Final loss is,

$$loss_t = -\log P(w_t*) + \lambda covloss_t$$

Intra-Attention on Decoder outputs

Traditional approaches attend only on the encoder states. But the current word being generated also depends upon what previous words were generated. So Paulus et al. [2017] used Intra-Attention on Decoder outputs. This approach helps avoids repeating particular word. Decoder context vector c_t^* is generated in a similar way to encoder attention. Finally, c_t^* passed on to generate $P_{vocab}(w)$.

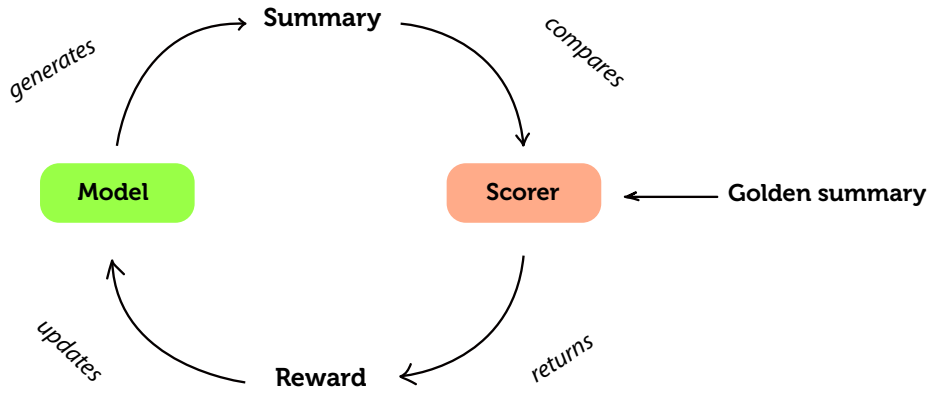
Learning from mistakes

A very big problem with the baseline summarization models is during the sampling stage. During training, we always feed in the correct inputs to the decoder, no matter what the output was at the previous step. So the problem that this gives rises is that the model doesn't learn to recover from its mistakes and assumes that it will be given the golden token at each step in the decoding. This works fine during training but during test time, we sample the next word from the output of the previous step, so if the model produces even one wrong word then the recovery is hard. A naive way to do rectify this problem is to toss a coin with $\mathbb{P}[heads] = p$, during decoding at training time and choose the token produced at previous step with probability p and the golden output token with probability $1 - p$. Now the network can't always assume that it'll be given the correct summary and hence learns to generalize better. In practice, this method gives only slight improvement but impacts convergence.But very recently Paulus et al. [2017] have come up with a Reinforcement Learning based training method that gave huge improvements.

Training using RL

The training in our baseline model is simply word-level Supervised Training. The model's aim is to output the reference summary, so we define a cross entropy loss between the target and the produced word. But this approach is fundamentally flawed. There are various ways in which the document can be effectively summarized. The reference summary is just one of those possible ways. Hence, the model's aim shouldn't be just restricted to outputting only the reference summary. There should be some scope for variations in the summary. This is the essential idea behind the Reinforcement Learning based training approach.

In this approach, during training, we first let the model generate a summary using its own decoder outputs as inputs. This is essentially sampling as described above. After the model produces its own summary, we evaluate the summary in comparison to the reference summary using the ROUGE metric. We then define a loss based on this score. If the score is high that means the summary is good and hence the loss should be less and vice-versa.



Policy Gradient-based Reinforcement Learning

We use a self-critical Policy gradient for training. We generate two strings y^s and $\hat{y}$.
$y^s \sim \mathbb{P}(y_t^s|y_1^s, \cdots, y_{t-1}^s, x)$ ie we are sampling from the decoder output distribution and $\hat{y}$ is generated by greedy search. Also let, y^* be the ground truth.
Let $r(y)$ is the reward for sequence y compared with $y*$. This could be calculated using any evaluation metric like ROUGE or a topic model. Finally we associate a loss as follows,

$$L_{rl} = (r(\hat{y}) - r(y^s)) \sum_t \log \mathbb{P}(y_t^s|y_1^s, \cdots, y_{t-1}^s, x)$$

This method gave astoundingly greater ROUGE scores over the previous state of the art results. But a big problem with this is rooted in the fundamental flaw in the ROUGE evaluation metric. it is possible to achieve a very high ROUGE score, without the summary being human readable. Particularly, it greatly improved recall. Now, since the above method optimizes for the ROUGE scores, it may produce summaries with very high ROGUE scores, but which are barely human-readable. So to curb this problem, we train our model in a mixed fashion using both Reinforcement learning and Supervised Training. We can interpret it as, RL training giving the summary a global sentence/summary level supervision and Supervised training giving a local word level supervision. So we need to use both cross entropy loss and this RL loss.

$$L_{mixed} = \gamma L_{rl} + (1 - \gamma) L_{ml}$$

8 Current Issues

Problem with the metric

ROUGE as a Metric is deficient. As pointed out by the Reinforcement learning method of Paulus et al. [2017], it is possible to achieve a very high ROUGE score, without the summary being human readable. This clearly means that the way ROUGE measures summary is different from how humans evaluate a summary.

Problem with the Dataset

A majority of the dataset that is available online is news dataset. A peculiar characteristic of these news datasets is that one can come up with a pretty good summary only by looking at the top few sentences. All the above-discussed models discussed above assume this and look at only the top 5-6 sentences of the source article. We need a richer dataset for multi-sentence Text Summarization. Also, another peculiarity of these models as with any other Deep Learning model is that they require huge amounts of data and computational power.

9 Future Work

To solve the problem of ROUGUE metric in the Reinforcement Learning based training method, we can instead learn a Discriminator separately first, which given a document and corresponding summary tells how good the summary it.

The problem of long document summarization has two main problems. First is the Vanishing Gradient Problem. Though LSTM's help a word at time step t pass along it's context along further in time arbitrarily, but the errors that we get a particular step aren't able to backpropagate further than 20-25 steps. So we are not able to learn the context properly in a long document. We propose to use a residual connection based architecture, to overcome this issue.

The motivation is that if we make residual connections between the time step t and $t - 2^i$ for all i, then the errors/gradients at a step can propagate back n steps in only $\log n$ steps. $\log n$ steps are short enough for the errors to back-propagate.

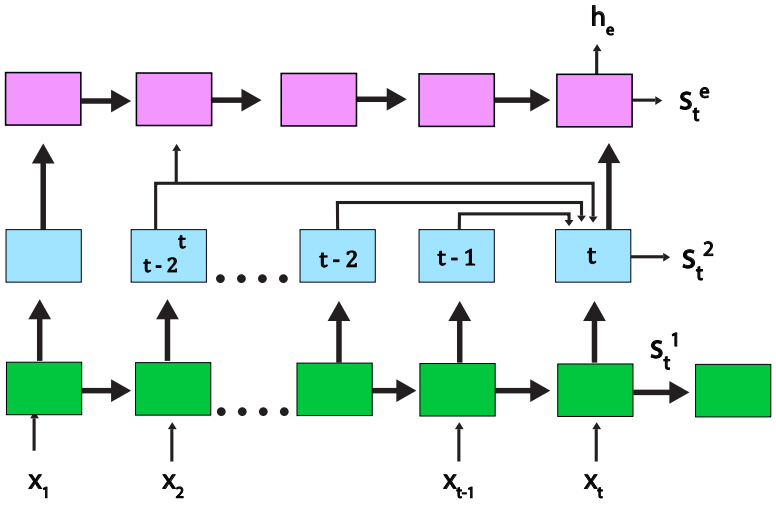


Figure 3: An image of stacked LSTM with residual connections.

Conclusion

This review shows that Deep Learning based approaches are promising and give some hope in solving Abstractive text summarization which had been largely unsolved till now. But the problems with the metric and lack of dataset are a challenge to scalability and generalizing to multi-sentence summarization.

References

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

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

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

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*, 2016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.