Abstractive Text Summarization
Using Seq2Seq Attention Models

Soumye Singhal   Anant Vats   Prof. Harish Karnick

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

25th April, 2018
Outline

The Problem

Deep Learning Approaches
   Encoder-Decoder with Attention
   Pointer-Generator Network
   Coverage Mechanism
   Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
   Residual Logarithmic LSTM
   Convolutional Seq2Seq to reduce training time
Why Text Summarization?

- Text Summarization is an important and hard problem towards understanding language.
- It would also give us the ability to process more information in a less time.
- Abstractive methods try to first understand the text and then rephrase it in short, using possibly different words.
- There are two main approaches that seriously try to tackle Abstractive Text Summarization
  - AMR-Representation Graph
  - Deep Learning based Seq2Seq Variants
Why Deep Learning?

- These approaches use Neural Seq2Seq Encoder-Decode Attention models.
- Inspired by the performance of Neural Attention Model in the closely related task of Machine Translation Rush et al. 2015 and Chopra et al. 2016 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.
- Since then a lot of work has been done to improve these models.
- Though they outperform other traditional abstractive techniques, they are plagued by various problems out of which not all have been solved in recent timing.
Outline

The Problem

Deep Learning Approaches
  Encoder-Decoder with Attention
  Pointer-Generator Network
  Coverage Mechanism
  Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
  Residual Logarithmic LSTM
  Convolutional Seq2Seq to reduce training time
Vanilla Encoder-Decoder

- It consists of an Encoder (Bidirectional LSTM) and a Decoder LSTM network.
- The final hidden state from the Encoder (thought vector) is passed into the Decoder.

1 Image taken from colah.github.io
Why do we need Attention?

- The basic encoder-decoder model fails to scale up.
- The main bottleneck is the fixed sized thought vector.
- 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.
- This is where attention comes it.
- It helps the model decide which part of the input encoding to focus on at each generation step to generate novel words.
- At each step, the decoder outputs hidden state $h_i$, from which we generate the output.
Attention is all you need!

- $importance_{it} = V \ast \tanh(e_i W_1 + h_t W_2 + b_{attn})$.
- Attention Distribution $a^t = \text{softmax}(importance_{it})$
- Context Vector $h^*_t = \sum_i e_i \ast a^t_i$

![Diagram of attention mechanism](https://talbaumel.github.io/attention/)

---

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

- Context Vector is then fed into two layers to generate distribution over the vocabulary from which we sample.
- \[ P_{vocab}(w) = \text{softmax}(V'(V[h_t, h^*_t] + b) + b') \]
- For the loss at time step \( t \), \( \text{loss}_t = -\log P(w_t^*) \), where \( w_t^* \) is the target summary word.
- \[ \text{LOSS} = \frac{\sum_{t=0}^{T} \text{loss}_t}{T} \]
- We then use the Backpropagation Algorithm to get the gradient and learn the parameters.
Generating the 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 choose the mode of the Distribution
▶ Sample from the distribution.
▶ **Beam Search** - Maintain $k$- the top hypothesis for the summary by generating from existing and pruning the result to retain top $k$. This ensures that each target word gets a fair shot at generating the summary.
Initial Problems

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

- They sometimes tend to reproduce factually incorrect details.
- Struggles with Out of Vocabulary (OOV) words.
- They are also a bit repetitive and focus on a word/phrase multiple times.
- Problems in generation.
Feature-rich Encoder and Hierarchical Attention (Nallapati et al. 2016)

- Input more information into encoder like word2vec, GloVe also linguistic features like POS Tags, TF-IDF, NE’s.
- Two Bi-Direction RNN at the source text
  - One at word level and another at the sentence level
  - Word level attention is then weighted by corresponding sentence level attention.

\[
P^a(j) = \frac{P^a_w(p)P^a_s(s(j))}{\sum_{k=1}^{N_d} P^a_w(k)P^a_s(s(k))}
\]
Outline

The Problem

Deep Learning Approaches

- Encoder-Decoder with Attention
- Pointer-Generator Network
- Coverage Mechanism
- Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems

- Residual Logarithmic LSTM
- Convolutional Seq2Seq to reduce training time
Pointer-Generator Network

Introduced by See et al. 2017.

- Helps to solve the challenge of OOV words and factual errors.
- Works better for multi-sentence summaries.
- Ides 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 the source text.
Pointer-Generator Network

Image taken from blog, www.abigailsee.com
At each step we calculate generation probability $p_{gen}$

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

$x_t$ is the decoder input.

Parameter $w_{h^*}, w_s, w_x, b_{ptr}$ are learnable.

Now this $p_{gen}$ is used as a switch.

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

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

The Problem

Deep Learning Approaches
- Encoder-Decoder with Attention
- Pointer-Generator Network
  - Coverage Mechanism
- Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
- Residual Logarithmic LSTM
- Convolutional Seq2Seq to reduce training time
Coverage Mechanism

- The cause of repetitiveness of the model can be accounted for by increased and continuous attention to a particular word.
- Coverage Vector $c^t = \sum_{t'=0}^{t-1} a^{t'}$
- By summing the attention we are keeping track of how much coverage each encoding, $e^i$ has received.
- Now, give this as input to attention mechanism.
- $importance_{it} = V \ast \tanh(e_iW_1 + h_tW_2 + W_c c_i^t + b_{attn})$
- $covloss_t = \sum_i min(a_i^t, c_i^t)$ penalizes overlap between attention at this step and coverage till now.
- $loss_t = -\log P(w_t^*) + \lambda covloss_t$
- Another Approach - Paulus et al. 2017 used Intra-Attention on Decoder outputs.
- Decoder context vector $c^*_t$ is generated in a similar way to encoder attention.
Outline

The Problem

Deep Learning Approaches
- Encoder-Decoder with Attention
- Pointer-Generator Network
- Coverage Mechanism
- Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
- Residual Logarithmic LSTM
- Convolutional Seq2Seq to reduce training time
Two major issues in Generation?

How to recover from mistakes?
▶ During training, we always feed in the correct inputs to the decoder, no matter what the output was at the previous step.
▶ Model doesn’t learn to recover from its mistakes.
▶ It assumes that it will be given the golden token at each step in the decoding.
▶ During testing if the model produces even one wrong word then the recovery is hard.

The summary is not Unique
▶ There are various ways in which the document can be effectively summarized. The reference summary is just one of those possible ways.
▶ There should be some scope for variations in the summary
Training using Reinforcement Learning

Introduced by Paulus et al. 2017 and is the current SOTA.

- First let the model generate a summary using its own decoder outputs as inputs.
- Evaluate the summary with the reference summary using ROUGE
- We then define a loss based on this score.
Fragility of the RL Method

- It’s possible to achieve a very high ROUGE score, without the summary being human readable.
- Reflects that ROUGE doesn’t exactly capture the way we humans evaluate summary.
- Now, since the above method optimizes for the ROUGE scores, it may produce summaries with very high ROUGE 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.

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

Metric

- Metrics like ROUGE (Lin 2004) and METEOR and BLUE
  - They are essentially string matching metrics
  - ROUGE-N measures the overlap of N-grams between the system and reference summary
  - ROUGE-L is based on longest common subsequences. Takes into account sentence level similarity.
  - ROUGE-S is the skip-gram variant

Problems

- ROUGE is not a natural way to evaluate summaries and is quite different from how humans evaluate summaries.
Dataset

Sentence level Datasets

- DUC-2004
- Gigaword

Large-Scale Dataset by Nallapati et al. 2016

- CNN/Daily Mail Dataset adapted for summarization.
- Contains roughly 200k news articles. CNN-Dailymail consists of passages, roughly of 10-15 lines. Each passage has associated with its handful of headlines which is treated as the summary.

PUBMED
Challenges

- As pointed out by Paulus et al. 2017, ROUGE as a metric is deficient.
- Dataset issues
  - A majority of the dataset that is available is news dataset.
  - lead-3 Baseline hard to beat.
  - Can come up with a 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.
- Scalability Issues - the Multi-sentence problem unsolved.
- Need a lot of data and computation power.
Outline

The Problem

Deep Learning Approaches
  Encoder-Decoder with Attention
  Pointer-Generator Network
  Coverage Mechanism
  Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
  Residual Logarithmic LSTM
  Convolutional Seq2Seq to reduce training time
Long Document Summarization

Deficiency of current models to handle long Sequences.

- Vanishing Gradient Problem is solved to some extent by LSTM which information passes along further.
- But for long sequences, the errors don’t propagate further back in time well. Around a maximum 20-25 steps only.
- For longer sequences, we need to context expanding potentially hundreds and thousands of time steps which normal LSTM simply can’t handle.
- Solution? - Residual Logarithmic LSTM’s
Residual Logarithmic LSTMs
Heuristics

- Effective backpropagation for longer sequences
- Inspired by skip connections in ResNet to pass along information.
- At each time step $t$, the passed context is not just from the previous state but also from all the $t - 2^j$ states for $j \leq \log t$.
- It allows us to pass the context for far back along the sequence in lesser number of connections. Basically $M$ time-step difference is covered in $\log M$ steps.
- This allows this model to be trained without the problems of vanishing gradient.
Implementation Details

- The basic model is a pointer-generator and coverage model with attention as introduced by See et al. 2017.
- The Recurrent Module is instead a Bi-directional Residual Logarithmic LSTM.
- Though usually these models require very big vocabulary due to the sheer volume of the data e.g. 200k, the pointer-generator gives a decent performance on over 50k vocabulary. We use \textbf{30k} vocabulary due to computational constraints.
- Embedding size was \textbf{256}.
  We observed that embeddings learned during training fared better than pre-trained embedding.
Implementation Details

- Hidden-Dimension - 128. SOTA papers use larger.
- Optimizer - Adam with 0.001 Learning rate
- Beam size of 4
- Dataset - CNN and Dailymail
- Implemented in Pytorch at https://github.com/soumye/Text-Summarization
- Maximum Article length - 300 words
- Maximum summary length - 110 words
Training

Due to the sheer complexity of the model, it is not trained in one step end to end. Instead, we **train it in multiple phases**. 

- Difficult to directly handle long sequences.
- Train on shorter Article and Abstract till the loss saturates. Then increase the sequence lengths
- Heuristically it is seen to perform better. The model incrementally handles the increasing sequence length. Training on long sequences from the start is just too much variance in data to handle.
Training

Conflict between pointer-generator and Coverage

▶ It was seen that the pointer-generator cross entropy loss and coverage loss interfered with each other and the loss didn’t decrease down much.

▶ So Initially the model is trained only on pointer-generator loss (ie $\lambda = 0$) using the incremental strategy. This was done of roughly 3-4 days till saturation.

▶ After that, the model is trained on the Coverage loss for roughly 3-4 hrs till saturation.
## Results and Benchmarks

The F-scores are as follows

<table>
<thead>
<tr>
<th>Model</th>
<th>ROUGE-1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>lead-3 baseline</td>
<td>39.2</td>
<td>15.7</td>
<td>35.5</td>
</tr>
<tr>
<td>Nallapati et al.</td>
<td>35.46</td>
<td>13.30</td>
<td>32.65</td>
</tr>
<tr>
<td>pointer-generator</td>
<td>36.44</td>
<td>15.66</td>
<td>33.42</td>
</tr>
<tr>
<td>pointer-generator + coverage</td>
<td>39.53</td>
<td>17.28</td>
<td>36.38</td>
</tr>
<tr>
<td>Residual-300(non-coverage)</td>
<td>29.2</td>
<td>11.5</td>
<td>27.3</td>
</tr>
<tr>
<td>Residual-300(coverage)</td>
<td>27.9</td>
<td>10.8</td>
<td>26.6</td>
</tr>
</tbody>
</table>
Analysing the Results

- Since we trained on CNN-Dailymail which can be summarized well only by looking at the first few sentences, it puts us at a disadvantage computing all these long-term dependencies using those skip connections. So, in essence, the previous models would work better as we see from the results.
- Due to computational constraints we needed to restrict the model parameters which limited its performance.
- Increasing training time might have helped though the loss seemed to saturate.
Analysing the Results

- For the multi-step process, model makes huge decreases in the loss in the early stages and then almost saturates down.
- In the early stages the output summary is very short but it is very decent. In the later stages, the summary is longer but there is a lot of repetitions.
- Using the pointer-mechanism makes our model a lot more extractive since the model doesn’t take many risks (more so during testing) and simply copies the word from the input.
Instability of Coverage loss?

- Before training on coverage loss the summary was essentially one or two novel sentences followed by repetitions of the same.
- After training on the coverage loss this problem was mitigated to a very limited extent.
- But, as can be seen in the results, for our model the coverage mechanism performs worse than the simple pointer-generator.
- Oscillating loss during training
- This might indicate that the coverage loss is unstable and we might need a more sophisticated method of curbing repeating attention which is much more aggravated in our Residual LSTMs due to the increased number of connections.
- More evidence and analysis is needed to support this claim.
Conclusions

- Not clear from our setting whether using Logarithmic LSTMs helped or not.
- Using more resources and a richer dataset like **PUBMED** might increase relative performance.
- There is also the scope of clubbing this with RL based training.
- The training time is humongous. For much longer documents it would scale up significantly.
- Not always feasible.
Outline

The Problem

Deep Learning Approaches
   Encoder-Decoder with Attention
   Pointer-Generator Network
   Coverage Mechanism
   Reinforcement Based Training

Metrics and Datasets

Daunting Challenges and Way Forward

Our Approaches to Solve these Problems
   Residual Logarithmic LSTM
   Convolutional Seq2Seq to reduce training time
To counter the various shortcomings of LSTMs, the way forward now seems to be Convolutional Neural Networks. They possess various qualities like:

- Parallel computation over its elements.
- Better GPU exploitation
- Easier optimization
- Thus decreasing training time by an order of magnitude.
Fairseq Model

4 Fairseq Conv-Net Architecture, courtesy: Facebook Research (fairseq), fairseq
Description of the Model

- **Convolutional Block Structure**
  - Input embeddings are convoluted (by blocks) by taking a fixed number of words together to get intermediate states.
  - Similar process is carried out on the decoder network.
  - Each block contains a 1D convolution and a non-linearity.
  - Blocks are stacked in layers to increase the field (the number of input nodes it depends on) of each output.
  - Gated Linear Units are used as non-linearity.
  - \( \nu([A \ B]) = A \otimes \sigma(B) \)
  - Residual connections are also added from input of each convolution to the output of the block.
Description of the Model

- **Multistep Attention**
  - Each decoder layer gets a separate attention mechanism.
  - Current decoder state is combined with previous target element to compute attention.
  - For $l^{th}$ decoder layer, attention is computed as dot product of decoder state summary and output of the last encoder block.
  - The input element embedding is also added to the encoder output before applying attention.
  - The attention parameters calculated in this step are implicitly available to future steps through the conditional inputs to the hidden states.
  - Thus this attention mechanism takes into account previously applied attention and behaves accordingly.
Description of the Model

- **Positional Embeddings**
  - Input elements are embedded into distributional space $\mathbb{R}^d$.
  - Absolute position of input elements is added to the embeddings to give the model a sense of order.

- **Normalization**
  - Weights are carefully initialized and parts of the network are scaled so as to keep variance throughout the network in check.
  - Gradients were also scaled based on the number of attention mechanisms used to stabilize learning.
Implementation Details and Training

- Both the encoder and the decoder consist of 512 hidden units.
- The dimensions of all the embeddings used is 512.
- Training algorithm used is Nesterov’s accelerated gradient method with momentum of 0.99.
- Gradients are re-normalized if their norm exceeds 0.1.
- Learning rate used was 0.25 which is decreased up to $10^{-4}$ during the course of training.
- Mini batch size is 64 sentences with restriction on max number of words in a mini batch.
## Results and Benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>ROUGE-1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>lead-3 baseline</td>
<td>39.2</td>
<td>15.7</td>
<td>35.5</td>
</tr>
<tr>
<td>Nallapati et al.</td>
<td>35.46</td>
<td>13.30</td>
<td>32.65</td>
</tr>
<tr>
<td>pointer-generator</td>
<td>36.44</td>
<td>15.66</td>
<td>33.42</td>
</tr>
<tr>
<td>pointer-generator + coverage</td>
<td>39.53</td>
<td>17.28</td>
<td>36.38</td>
</tr>
<tr>
<td>Residual-300(non-coverage)</td>
<td>29.2</td>
<td>11.5</td>
<td>27.3</td>
</tr>
<tr>
<td>Residual-300(coverage)</td>
<td>27.9</td>
<td>10.8</td>
<td>26.6</td>
</tr>
<tr>
<td>Conv Seq2Seq</td>
<td>27.46</td>
<td>10.78</td>
<td>25.82</td>
</tr>
</tbody>
</table>
Limitations of Deep Learning

- This work clearly highlights some of the major problems in these deep learning based approaches to NLP.
- End to End models seems too magical to be true. Need **Modularity**
- Need good intermediate representation on which these models can work.
- Is **AMR** the key?
References


