swissQuant
Mastering Complexity

Mathematical Challenge March 2018

Abstractive, multi-document, query-related text summarization

References

1. Aggarwal, C. & Zhai, C. (2012), Mining Text Data, Springer-Verlag New York Inc .
2. Young, T.; Hazarika, D.; Poria, S. & Cambria, E. (2017), 'Recent Trends in Deep Learning Based Natural Language Processing.', CoRR abs/1708.02709
3. Hermann, K. M.; Kociský, T.; Grefenstette, E.; Espeholt, L.; Kay, W.; Suleyman, M. & Blunsom, P. (2015), Teaching Machines to Read and Comprehend., in Corinna Cortes; Neil D. Lawrence; Daniel D. Lee; Masashi Sugiyama & Roman Garnett, ed., 'NIPS' , pp. 1693-1701 .
4. See, A.; Liu, P. J. & Manning, C. D. (2017), 'Get To The Point: Summarization with Pointer-Generator Networks.', CoRR abs/1704.04368
5. Baumel, T.; Eyal, M.; Elhadad, M. (2018), 'Query Focused Abstractive Summarization: Incorporating Query Relevance, Multi-Document Coverage, and Summary Length Constraints into seq2seq Models'
6. McDonald, R. T. (2007), A Study of Global Inference Algorithms in Multi-document Summarization., in Giambattista Amati; Claudio Carpineto & Giovanni Romano, ed., 'ECIR' , Springer, , pp. 557-564 .

Description

Automatic summarization is the process of shortening a text document with software, in order to create a summary with the major points of the original document, i.e. find a subset of data which contains the 'information' of the entire set.

Summarization techniques can roughly be divided into two large sub-groups according to the input and output type [1]. Input can either consist of a single document or a cluster of related documents (multi-document summarization). Output of a summarization engine can be extractive or abstractive. Extractive summarizers identify the most important sentences in the input (conveying key information) and string them together to form a summary. Abstractive summarization involves paraphrasing sections of the source input, i.e. identifying key concepts and forming understandable sentences conveying the key information.

The extractive approach is easier, because copying large chunks of text from the source document ensures baseline levels of grammaticality and accuracy. On the other hand, sophisticated abilities that are crucial to high quality summarization, such as paraphrasing, generalization, or the incorporation of real-world knowledge, are possible only in an abstractive framework.

Due to the difficulty of abstractive summarization, the great majority of past work has been extractive. However, recent developments in neural-attention based sequence-to-sequence models (see [2]) have led to the state-of-the-art results on the task of abstractive single document summarization and development of large annotated data sets for the respective task.

Here, we will focus on extensions of the model presented in [4], Pointer-generator networks for generic single-document summarization, namely in the direction of multi-document and query-focused context.

Sequence-to-sequence attentional model

The task of summarization can be cast as a sequence-to-sequence learning problem, where the input is original text and the output is its summarized version. A sequence-to-sequence (seq2seq) model typically contains following sub-units:

- Encoder
- Decoder
- Attention mechanism

Encoder transforms input text (unstructured data) into numeric format. This procedure is usually referred to as vectorization, word embedding or encoding. State-of-the-art encoders are mostly variations of recurrent neural networks (RNN). A recurrent neural network (RNN) is a class of neural networks where connections between units form a directed graph along a sequence. This allows it to capture context for word or character sequences. A variant of RNN especially successful in machine translation and language generation are so called Long short-term memory (LSTM) networks. LSTM units are building nodes for layers of a RNN, commonly composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates is a neuron; as they compute an activation of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM. In [4], they used RNN structures with bi-directional LSTM cells to encode input text [1].

Decoder [5] is a neural network unit that generates the next word in the summary conditioned on the representation of the prefix of the generated text and a dense context vector representing the input sequence. The decoder is also commonly implemented by an RNN, with a softmax layer that turns a vector into a distribution over the vocabulary. In [4], they used a single-layer unidirectional LSTM.

Attention mechanism [5] is a unit that determines the importance of each encoded word at each decoding step, and maps the variable length list of encoded words representations into a *fixed-size context representation*. The attention mechanism is commonly implemented using multiple levels of fully connected layers with a softmax layer to normalize attention weights.

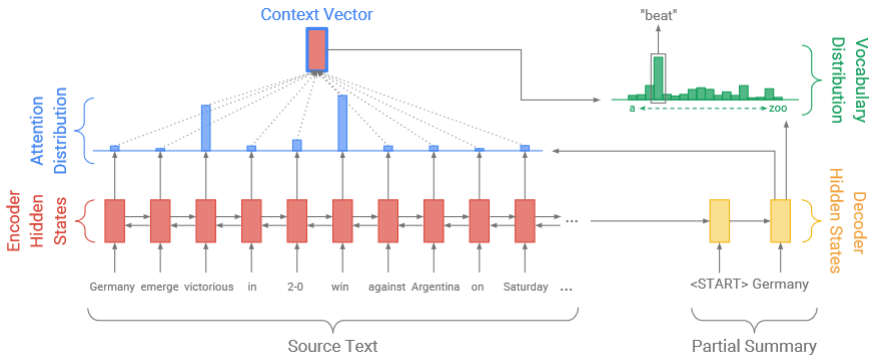

Figure 1 Sequence-to-sequence model with attention

The sequence-to-sequence attentional model is depicted on figure 1.

[1] Bi-directional networks allow for context capturing from both preceding and following words, which is useful, as linguistic constructions are not necessarily unidirectional in nature.

The attention distribution a^t [4] in is calculated as:

$$e_i^t = \gamma^T \tanh(W_h h_i + W_s s_t + b_{attn})$$
$$a^t = softmax(e^t)$$

Where $\gamma, W_h, W_s, b_{attn}$ are learnable weights.

The attention distribution is then used to produce a weighted sum of the encoder hidden states (context vector) h_t^*:

$$h_t^* = \sum a_i^t h_i$$

The context vector serves as a fixed-sized interpretation of input at step t. This is then concatenated with the decoder state s_t and fed through two linear layers to generate the vocabulary distribution:

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

Where V, V', b, b' are again learnable. If we marginalize P_{vocab}, we can derive final distribution from which we can predict the next word in the output sequence:

$$P(w) = P_{vocab}(w)$$

During training, the following loss function at each step t is the negative log-likelihood for the target word w_t^*:

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

Such networks can be trained on large annotated corpora in form of question-answer pairs, such as the CNN/Daily corpus [3]. More details about model training can be found in [4].

Pointer-generator network

Classical seq2seq models sometimes tend to produce factually incorrect summaries [4]. To tackle this problem, one can incorporate a pointer mechanism to aid a more accurate reproduction of information. A pointer-generator mechanism allows us to handle out-of-vocabulary words and output dictionaries depending on input seq. dimension, while retaining the ability to generate new words.

The pointer generator network in [4] is a hybrid between the seq2seq and pointer network models, as it allows both copying words via pointing, and generating words from a fixed vocabulary. Once the attention distribution and context vector have been computed, we can compute the generation probability:

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

Where w_h^*, w_s, w_x, b_{ptr} are learnable, σ denotes a sigmoid.

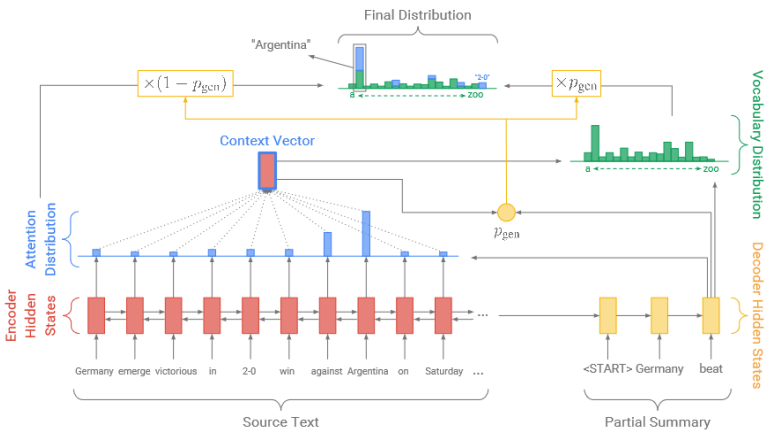

Figure 2 Pointer generator network

p_{gen} is used as a soft switch 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 can derive the following probability distribution over the extended vocabulary:

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

Coverage mechanism

Repetition of tokens when generating the output is a common problem for seq2seq models [4], especially pronounced when generating multi-sentence text. To tackle this problem, one can add a coverage mechanism to keep track of what has been summarized already. In [4], this was done using a coverage vector c^t, a sum of attention distributions of previously generated words:

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

This vector is an unnormalized distribution over source document words that shows the degree of coverage these words have received in the summary generated up to step t. The coverage vector can then be used as an additional input to previously derived attention mechanism:

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

This ensures that the attention mechanism's current decision is informed by a reminder of its previous decisions (summarized in c^t).

Additionally, as pointed out in [4], one needs to add the coverage loss to the objective function to penalize the overlap between each attention distribution and the coverage so far:

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

Questions

- *The approach described in [4] can be used in single-document, generic abstractive summarization setting. Investigate potential expansions of the approach to a multi-document, query-focused setting in available literature (such as [5]).*
- *In [6], a global inference integer linear program is suggested as a way to generate multi-document, query-focused summary. Find a way to combine this approach with abstractive summarization algorithm presented in [4] to obtain a multi-document abstractive, query-focused summarization engine.*
- *Benchmark the developed model against methods presented in the literature using a well known publicly available data set for text summarization, like the CNN/Daily dataset [3] or DUC 2005 onwards (http://duc.nist.gov/data.html).*

We look forward to your opinions and insights.

Best Quant Regards,

swissQuant Group Leadership Team