Natural Language Understanding, Generation, and Machine Translation (2020–21)

School of Informatics, University of Edinburgh
Mirella Lapata

Tutorial 5: Summarization (Week 10)

In class we have seen how to apply the encoder-decoder model to the task of abstractive single document summarization. In this tutorial we will discuss extractive summarization and the challenges associated with it.

Question 1:

Extractive summarization, as the name implies, creates a summary by extracting sentences from the source (i.e., input) document. Most extractive models frame summarization as a classification problem. Given document D consisting of a sequence of sentences $\{s_1 \ldots, s_m\}$, we aim to select a summary from D by selecting a subset of j sentences where $j < m$. We do this by scoring each sentence within D and predicting a label $y_L \in \{0, 1\}$ indicating whether the sentence should be included in the summary.

a. We want to apply supervised training, our objective is to maximize the likelihood of all sentence labels $\mathbf{y_L} = (y_L^1, \ldots, y_L^m)$ given the input document D and model parameters θ. Can you write down this objective?

$$\log p(\mathbf{y}_L|D;\theta) = \sum_{i=1}^{m} \log p(y_L^i|D;\theta) \tag{1}$$

b. In order to classify each document sentence as being part of the summary or not, we need to find a way to represent D in a feature space. We will use a neural network-based document encoder. Can you think of how to build this encoder in a **hierarchical** fashion? How would you represent sentences, and then the document?

A hierarchical document encoder derives the document meaning representation from its sentences and their constituent words. We can use RNNs to encode the sentences or CNNs. For example, we can obtain continuous representations of sentences by applying single-layer convolutional neural networks over sequences of word embeddings and then rely on a recurrent neural network to compose sequence of sentences to get document embeddings. Alternatively, we can use a bi-directional RNN at the word level, to compute the hidden state representation at each word position sequentially (forward and backward), based on the current word embeddings and the previous hidden state. The bidirectional RNN would give two hidden representations which we could concatenate. To obtain a document representation D, we then use average pooling of the concatenated hidden states of the bi-directional sentence-level RRN:

$$\mathbf{d} = \tanh(W_d \frac{1}{N_d}) \sum_{j=1}^{N^d} [\mathbf{h}_j^f, \mathbf{h}_j^b] + \mathbf{b}$$

bidirectional embedding concatenate

where $\mathbf{h}_j^f$ and $\mathbf{h}_j^b$ are the hidden states corresponding to the jth sentence of the forward and backward sentence-level RNNs respectively, N_d is the number of sentences in the document and $[,]$ represents vector concatenation.

c. So far we have managed to obtain feature representations for documents and sentences. But how do we extract sentences to create a summary? Our summarizer should label each document sentence with label 1 or 0 by estimating its **relevance** within the document. Can you think of ways to model $p(y_L|D)$?

$$p(y_t|s_t, D) = \text{softmax}(g(h_t, h_t^{'}))$$

$$g(h_t, h_t^{'}) = U_o(V_h h_t + W_h^{'} h_t^{'})$$

$$h_t = \text{LSTM}(s_t, h_{t-1})$$

$$h_t^{'} = \sum_{i=1}^{p} \alpha_{(t,i)} h_i$$

$$\text{where } \alpha_{(t,i)} = \exp \frac{(h_t h_i)}{\sum_j \exp(h_t h_j)} and \text{p=t-1}$$

where $g(\cdot)$ is a single-layer neural network with parameters U_o, V_h, and $W_h^{'}$. h_t is an intermediate RNN state at time step t. The dynamic context vector $h_t^{'}$ is essentially the weighted sum of the encoder hidden states, seen so far. An alternative solution would be to use a BiLSTM and take all the attended representations of all sentence or/and take into account the probabilities of previous sentences to belong to the summary.

d. Once you have a model that labels each sentence as positive/negative how do you create a summary?

We rank sentences in the document D by $p(1|s_t, D, \theta)$, the confidence scores assigned by the softmax layer of the sentence extractor and generate a summary by assembling together the m best ranked sentences.

Question 2:

How would you train the above model? In order to train our extractive model, we need ground truth in the form of sentence-level binary labels for each document, representing their membership in the summary. However, most summarization corpora only contain human written abstractive summaries as ground truth. How could you obtain binary labels from existing corpora, e.g., from CNN/Daily Mail?

To solve this problem, we use an unsupervised approach to convert the abstractive summaries to extractive labels. Our approach is based on the idea that the selected sentences from the document should be the ones that maximize the ROUGE score with respect to gold summaries. Since it is computationally expensive to find a globally optimal subset of sentences that maximizes the Rouge score, we employ a greedy approach, where we add one sentence at a time incrementally to the summary, such that the ROUGE score of the current set of selected sentences is maximized with respect to the entire gold summary. We stop when none of the remaining candidate sentences improves the ROUGE score upon addition to the current summary set. We return this subset of sentences as the extractive ground-truth, which is used to train our RNN based sequence classifier.

Question 3:

Can you think of why extracts might be preferable to abstracts? Can you also think of reasons why extracts might make a bad summary?

Extractive summaries will not hallucinate, the sentences of the original document are

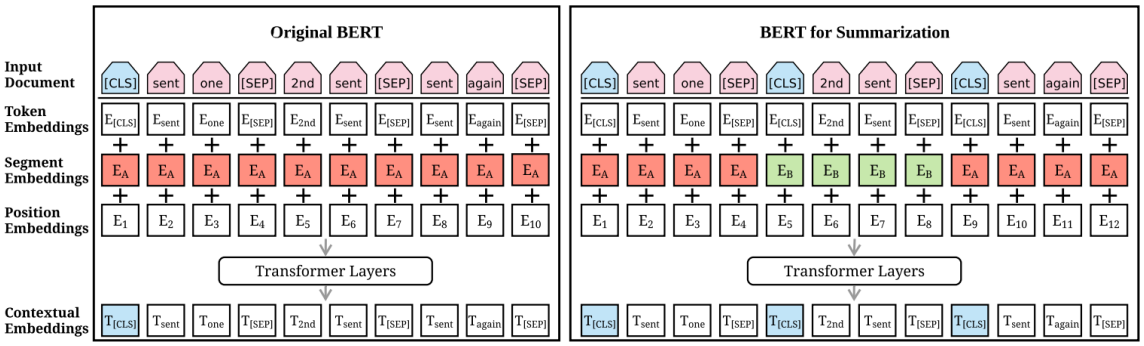

Figure 1: Architecture of the original BERT model (left) and new encoder (right). The sequence on top is the input document, followed by the summation of three kinds of embeddings for each token. The summed vectors are used as input embeddings to several bidirectional Transformer layers, generating contextual vectors for each token. We extend BERT by inserting multiple [CLS] symbols to learn sentence representations and using interval segmentation embeddings (illustrated in red and green color) to distinguish multiple sentences.

not altered in any way. They also avoid repetitions, UNK tokens, and can exploit positional biases (e.g., extract sentences from the beginning of the document) to create good enough summaries. The problem with extracts is that there is no way to modulate the sentence length, and they tend to generally have long sentences. Finally, since sentences are cut and pasted from the document there is no way to make sure that they are thematically related, so the summary may be locally (and globally incoherent).

Question 4:

Pretrained language models have been employed as encoders for sentence- and paragraph-level natural language understanding problems involving various classification tasks (e.g., predicting whether any two sentences are in an entailment relationship; or determining the completion of a sentence among four alternative sentences). Can you think of how you would use a pretrained language model like BERT to encode documents for extractive summarization?

Here just give them an intuition about how this might work. The students should be familiar with BERT, but perhaps revise with the aid of the figure above. Since BERT is trained as a masked-language model, the output vectors are grounded to tokens instead of sentences, while in extractive summarization, most models manipulate sentence-level representations. Although segment embeddings represent different sentences in BERT, they only apply to sentence-pair inputs, while in summarization we must encode and manipulate multi-sentential inputs

In order to represent individual sentences, we insert external [CLS] tokens at the start of each sentence, and each [CLS] symbol collects features for the sentence preceding it. We also use interval segment embeddings to distinguish multiple sentences within a document. For $sent_i$ we assign segment embedding E_A or E_B depending on whether i is odd or even. For example, for document $[sent_1, sent_2, sent_3, sent_4, sent_5]$, we would assign embeddings $[E_A, E_B, E_A, E_B, E_A]$.This way, document representations are learned hierarchically where lower Transformer layers represent adjacent sentences, while higher layers, in combination with self-attention represent multi-sentence discourse. A schematic representation is shown in Figure 1.

Let vector t_i denote the vector of the i-th [CLS] symbol and the representation for $sent_i$. Several inter-sentence Transformer layers are then stacked on top of BERT outputs, to capture document-level features for extracting summaries:

$$\tilde{h}^l = \mathrm{LN}(h^{l-1} + \mathrm{MHAtt}(h^{l-1})) \tag{2}$$

$$h^l = \mathrm{LN}(\tilde{h}^l + \mathrm{FFN}(\tilde{h}^l)) \tag{3}$$

where $h^0 = \mathrm{PosEmb}(T)$; T denotes the sentence vectors output by the encoder, and function PosEmb adds sinusoid positional embeddings to T, indicating the position of each sentence. The final output layer is a sigmoid classifier:

$$\hat{y}_i = \sigma(W_o h_i^L + b_o) \tag{4}$$

where h_i^L is the vector for $sent_i$ from the top layer (the L-th layer) of the Transformer.

The loss of the model is the binary classification entropy of prediction $\hat{y}_i$ against gold label y_i. Inter-sentence Transformer layers can be then jointly fine-tuned with the BERT-based encoder.