Extractive and abstractive summarization

Moreno D'Incà (223820)
University of Trento
Via Sommarive, 9, 38123 Povo,Trento TN
moreno.dinca@studenti.unitn.it

Abstract

Text summarization is becoming more and more nedded to help people saving time. In this work both extractive and abstractive summarization are exploited using the CNN daily mail dataset. Extractive summarization aims at finding the best subset of sentences from the original article which best represents it, abstractive, instead, generates a totally new text as summary. The main architecture leverages is the seq2seq model, in the first scenario the model is a modified implementation of (Nallapati et al., 2016); for abstractive summarization the model is an adaptation of the Pytorch tutorial (Robertson,). The performances of both models have margins of improvement, still they are a good starting point for future works. The extractive summarization notebook is available here: Open in Colab, the abstractive one here: Open in Colab.

1 Introduction

Summarization is the task of generating a text (summary) which condense the main points of a given article. Nowadays the concept of summarization becomes really useful in many areas, whenever a text is too long and a summary is needed, an automatic summarization model can help and save time. There are two main methods for summarization: extractive and abstractive. Extractive summarization is the easiest one, it consists in summarizing the original article using original sentences, extracting the summary directly from the original text. This method can be compared to highlighting. Abstractive summarization, on the other hand, aims at generating a summary with new sentences which are not used in the original article, here the main challenge is to generate a fluent summary maintaining an high level of knowledge from the original article. In the literature there exist multiple techniques to face this problem, from scoring methods to deep learning models. Scoring methods are mainly used for extractive summarization, for instance Raju et al. (Raju and Allarpu, 2017) describes a Sentence Scoring Method to score the sentences of the original article, the final summary will be the top ranked sentences. Another method for extractive summarization is to encode the meaning of each sentence and the whole article to then use a classification model (encoder) which has to predict whether a specific sentence has to appear in the final summary (Nallapati et al., 2016). The encoder-decoder model can also be implemented for abstractive summarization. Here one of the most popular is the seq2seq approach where two RNNs are used as encoder and decoder. This method was initially introduced by Google for machine translation, then spread to other fields including summarization. The seq2seq architecture takes in input a sequence of words (or sentences) and it outputs another sequence based on the problem to overcome (e.g. for machine translation the traslation of the input sequence, for summarization the summary), the model is composed by an encoder which encodes the original sequence and the decoder which has to generate the new sequence (Sutskever et al., 2014).
This project aims at solving the summarization problem using both extractive and abstractive methods, first the problem is more formally described, then the methods used are described along with the final results.

2 Problem statement

2.1 Extractive summarization

Given in input the set of sentences $< S0, S1, ..., Sn >$ of the article to summarize, extractive summarization aims at finding the subset of sentences $< s0, s1, ..., sm >$ which best represent its meaning. In this setting the performances are measured by means of Rouge scores, recall, precision and accuracy. The vocabulary is used to encode the words of the original articles.

2.2 Abstractive summarization

In this approach the article is summarized by generating a text with new words taken from a predefined vocabulary which has to be set a priori and it gives the model the knowledge to understand the articles and to generate the summary. Here the performance metric is Rouge scores.

3 Data Analysis

The dataset used in this project is the CNN Dailymail. There are different versions for both extractive and abstractive summarization. For the first setting the provided dataset from the Professor has been used (ext,), this dataset is already split in training, validation and test sets, all of which contain the articles and the summaries. Each summary is composed by a binary vector representing each sentence. The i-th position is 1 if the i-th sentence of the article appears in the final summary, otherwise 0. This dataset is composed by $286,817$ articles for the training set, $13,368$ samples for the validation and

$11,487$ pairs for the test set; due to processing capabilities a subset of the training set has been used (5k articles). In the abstractive setting the dataset used is taken from Huggin Face (abs,), apart for the fact that this dataset is abstractive, the choice was also made for the simplicity of importing the data (using the datasets module). Here the size of the sets is the following: training $287,113$, validation $13,368$ and test set $11,490$; each sample contains the article to summarize and the abstractive summary which the model should generate. The articles has been preprocessed in order to remove stop words, punctuation and symbols, this preprocessing has not been applied to the summaries in order to let the model generate fluent and human understandable summaries.
In both datasets a vocabulary for the hot encoding of the articles is needed. In the extractive setting the vocabulary has been taken from a github repository containing the processed CNN Dailymail dataset (pro,); for the abstractive one the vocabulary is created during the processing of the dataset, each new word will have a specific hot encoding used to represent it. Since the vocabulary size is strictly related to the training time via the complexity of the models (number of parameters) through the embedding layer (and for the abstraction even the output layer), both versions of the vocabulary can have a maximum size which can be set a priori by the user, during the experiments the size will be set to 50K as described in (See et al., 2017). There is also a need to encode 4 specific tokens: the unknown token ([UNK]) used to represent an OOV word, the padding ([PAD]) token used to pad the articles if their length is lower then the specific length (set to 400) and the decoder start and stop tokens ([START], [STOP]) used from the decoder to know when to start decoding and to know when a summary is finished.

4 Models

The model architecture used in this project is mainly based on the seq2seq method for both extractive and abstractive summarization, this model is composed by two main components, the encoder and the decoder. The encoder has the objective to encode the input in order to embeds the meaning of the input article in a vector (also referred as the context vector) this vector is then used by the decoder to generate the summary.
In the literature the encoder and decoder are mainly Recurrent Neural Networks (RNNs) but other architectures have been tried, for example Convolutional Neural Networks (CNNs) have shown good results (Gehring et al., 2017). This work will focus on the RNNs architecture since they still have the focus for this area of research.

4.1 Extractive summarization

The model implemented for this setting is a re-implementation of the paper (Nallapati et al., 2016) with some modifications. In this approach each article is split in sentences, the model takes in input a sentence at a time, the encoder encodes this sentence and it outputs its embedding (context vector) and the final hidden state of the Gated Recurrent Units (GRU), then, for each sentence, the decoder has to classify if it belongs to the summary or not. The final summary will be the set of sentences classified as 1.
The econder is made of two main GRU layers, the first one to capture the meaning at word level and the second one at sentence level. The layers are bidirectional to capture both directions: sentences and article. The encoder starts to encode each word sequentially generating one hidden state representation for each word, these hidden states are concatenated and used as input to the second GRU layer along with the hidden state from the previous sentence, this will generate the hidden state representation for each sentence. At this point the embedding of the entire document is computed as (Nallapati et al., 2016):

$$H = \frac{1}{N_d}\sum_{j=1}^{N_d}[h_j^f, h_j^b] \tag{1}$$

$$d = tanh(W_d H + b) \tag{2}$$

Where N_d is the number of sentences in the article and $[h_j^f, h_j^b]$ is the concatenation of the forward and backward hidden states for the $j-th$ sentence. In the code the encoder outputs directly the concatenation of the hidden states ($[h_j^f, h_j^b]$), these are then summed and normalized (1), this object is then one of the inputs to the decoder.
The decoder iterates over all the sentences, it takes in input the current sentence hidden state, H and the current summary state to compute (Nallapati et al., 2016):

$$content = W_c h_j \tag{3}$$

$$saliency = h_j^T W_s d \tag{4}$$

$$novelty = h_j^T W_r tanh(s_j) \tag{5}$$

$$x = content + saliency - novelty \tag{6}$$

$$P(y_j|h_j, s_j, d) = \sigma(xA^T + b) \tag{7}$$

Where:

- h_j is a non-linear transformation of the concatenation of the $j-th$ sentence hidden states.
- d is (2) implemented with a Linear layer and a tanh activation function.
- s_j is the summary state at $j-th$ sentence.
- y_j is the binary probability of being or not part of the sammary for the $j-th$ sentence

The current summary state (s_j) is initialized to zeros, then it is the sum over the sentence hidden states weighted

with the probability of being part of the summary (Nallapati et al., 2016):

$$s_j = \sum_{i=1}^{j-1} h_i P(y_i = 1 | h_i, s_i, d) \quad (8)$$

The content (3) represents the content information of the sentence, it is implemented via a Linear layer without bias, the saliency (4) represents how much importance the current sentence is with respect to the article, it is implemented via a bilinear layer without bias, and the novelty (5) represents the redundancy of the sentence with respect the current summary state, also this one implemented via a bilinear layer (Nallapati et al., 2016). The non-linear transformation of h_j is implemented via a Linear layer with tanh as non linear activation function.
These three objects are the input to the final classifier which will predict the probability of being part of the summary or not for the $j-th$ sentence. Based on analysis on the dataset the maximum number of sentences is set to 50 and the maximum number of tokens per sentence is set to 30.

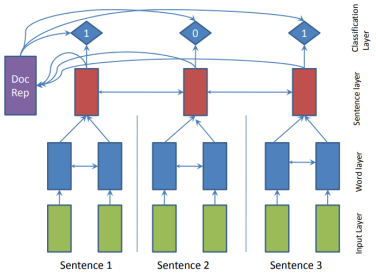

Figure 1: A simplification of the overall system from (Nallapati et al., 2016)

4.2 Abstractive summarization

The model implemented for this setting is an adaptation of the seq2seq model for translation explained in the pytorch tutorials (Robertson,), it is composed by the encoder and decoder with attention mechanism. The encoder has the objective to generate an embedding representation of the article, similarly to the extractive approach. It is composed by an Embedding layer and a GRU layer, it takes in input one word at a time and it outputs the context vector and the hidden state from the GRU layer. The context vector of each word is concatenated and used by the attention on the decoder, instead the final hidden state is used as initial hidden state for the decoder. The decoder is composed by the Embedding layer, the GRU layer to capture the sequence meaning and three Linear layers: two for the attention and one for the final output. The decoder generates one word of the summary at a time, taking in input the previous generated word (initially the [START] token), the current hidden state (initially the last hidden state from the encoder) and the context vector from the encoder. At training time teacher forcing is leveraged to improve the learning of the model: with probability p (set to 0.4) the model will take in input the true previous word from the real summary (instead of the generated one), this is supposed to speed up the learning and improve the final overall performances (still p has to be low, otherwise the model will not generate fluent summaries at testing time). The previous word (W^{prev}) is embedded and used to compute the attention weights and the attention itself. The weights are computed using a Linear layer with softmax as activation function inputting the concatenation of the embedded word (W^{prev}) with its hidden state (W_h^{prev}), this will produce the weights assigned to each word of the article, these weights are then applied to the encoder output (article representation) via a batch matrix-matrix product (BMM) to create the final attention. This attention is then concatenated to the initial embedding of the previous word (W^{prev}) and applied to a Linear layer, its output is applied to the GRU layer with the hidden state from the previous word (W_h^{prev}) and then a final Linear layer is applied. The decoder returns the output representing the generated word, the final hidden state which will be used for the next iteration and the current attention (which can be used for debugging purposes).

In this setting the articles are truncated to 400 tokens, if the length of an article is lower a set of padding tokens are applied, the same approach is used for the summaries where the maximum length is 100 tokens. The summaries are bounded by a [START] token and a [STOP] token, these tokens delimit the summary (the padding tokens are applied outside the [STOP] token).

5 Experiments

Several experiments have been performed to improve the overall performances for both extractive and abstractive summarization.

5.1 Extractive

Multiple losses have been leverage for this approach: cross entropy loss, negative log likelihood (NLL) loss and mean squared error loss. After several runs to assess the goodness of the losses the cross entropy loss shown the overall best performance. The optimizer exploited is Adam; during the experiments different learning rates have been tried $[1e-1, 1e-2, 1e-3, 1e-4, 1e-5]$, $1e-3$ shown to be the most stable one. Since, in this setting, a lot of targets are zeros (only very few sentences are selected for the summary) the model is inclined to be satisfied in predicting only zeros, for this reason multiple experiments have been run to assess the best weight to assign to each class. The cross entropy loss allows to weight the classes in order to differentiate the importance of each class (normally the weights are set to 1). The weight for the true (one) class (being part of the sum-

mary) is maintained to 1, for the false (0) class (not being part of the summary) multiple weights have been tried, if the weight is too low the model will predict always ones, instead if the weight is too high the model will behave as previously specified. The weight with best trade off has been 0.1. The metrics used during the experiments have been the Rouge score, recall, precision and accuracy scores. The Rouge score encodes the goodness of the generated summary with respect to the real one, the recall, precision and accuracy give a clue about the performances of the model with respect to the targets. Since the model still had difficulty in predicting the correct sentences, a new loss function have been applied:

$$CE(pred, trg, weight = w) + (1 - R * P)^2 \quad (9)$$

Where CE is the cross entropy loss, pred are the predicted summaries, trg the real summaries, w the weights of the classes and R, P the recall and precision scores from the previous epoch (initially 0). The idea is to reward the model if both recall and precision are high, this means that both ones and zeros are correctly classified, if this is not the case the loss function will increase.
Another experiment run has been to change the output of the model in order to output just one probability (given a sentence, the probability of being part of the summary) and to have a threshold set to 0.5 (if the output is higher the sentence will be part of the generated summary). This approach has been implemented along side with the mean squared error loss, since this loss does not take in account the weights, new losses have been implemented to avoid this fact, however this setting did lead to worst performances, therefore the efforts were diverted to the first approach.
In order to understand the quality of the results and to assess the integrity of the model, a random approach has been run to create a baseline for reference. The approach selects randomly four sentences for each article to create a random summary, the final results of this baseline has been averaged over ten runs in order to have a more stable and valid outcome, the model should outperform this baseline to be helpful.

5.2 Abstractive

In this setting the loss function is the negative log likelihood following (Robertson,) with Stochastic Gradient Discent (SGD) as optimizer. The learning rate, after fine tuning it, has been set to 0.01. The loss function has been set to ignore the PAD token, in this way this token will not be learnt during training. The main problem encountered in this approach is the bad quality of the results (the model is really slow at learning). In order to assess the goodness of the network, the model has been trained with a very small subset of the training set (1 article and 10 articles) in order to understand if it is able to overfit the set. The model was able to overfit it learning the summaries, an example can be seen in Table 4. After this experiment the model has been trained for several thousands epochs with a subset of the training set (5k articles), this size is related to the training time and the limitation of colab (a higher size would have been prohibitive). The model still shown a low rate of learning, for this reason other experiments have been run to fine tune the learning rate and the optimizer, the final system is still slow with margin of improvement (mostly in terms of accuracy).
In order to improve the accuracy another experiment has been to rely on word2vec for the embeddings of the words, in this setting there is no need for a vocabulary, each word is embedded using word2vec, the embedding is then used as input to the abstractive model (without the embedding layer), the model then outputs an embedding vector, the translation to a word is performed using the $most\_similar$ function provided by word2vec. This approach, however, did not lead to improvement therefore it has been abandoned.

6 Results and conclusions

The results of the models on both settings have margins of improvement, specially for the abstraction approach. The extractive summarization, after 90 epcohs of training, reached the results shown in Table 1, compared to the baseline performances of the random approach (Table 2), the model is performing better.

	VALIDATION			TEST		
	Recall	**Precision**	**f1**	**Recall**	**Precision**	**f1**
Rouge-1	0.54	0.47	0.48	0.55	0.46	0.47
Rouge-2	0.40	0.33	0.33	0.40	0.32	0.33
Rouge-L	0.52	0.44	0.45	0.52	0.44	0.45

Table 1: Extractive perfomances on validation and test sets

	VALIDATION			TEST		
	Recall	**Precision**	**f1**	**Recall**	**Precision**	**f1**
Rouge-1	0.27	0.30	0.27	0.27	0.30	0.27
Rouge-2	0.11	0.13	0.11	0.11	0.13	0.11
Rouge-L	0.24	0.27	0.24	0.24	0.27	0.24

Table 2: Baseline perfomances on validation and test sets

These high results could be misleading, in fact during the experiments I noticed that the model is inclined to predict always the same subset of sentences (mostly the first sentences of the articles) with little regard about the meaning and fluency of the summary, this behavior should be avoided, some experiments have been done to remove this deficit, the model shown to be really sensitive about the weight assigned to the class 0 on the loss function, this hyperparameter has been fine tuned, but better values could be found.
The results of the abstractive summarization are quite low (Table 3), since the model is able to overfit small training sets the bad performances should not be due to the model. After some though and research the most plausible conclusion is the fact that the low number of epochs

($\sim$ 3500) and the small subset of the training set used to train the model (5k articles) is not enough leading the system to underfitting. For instance in (Robertson,) the model is trained with 75k epochs and the whole training set. To conclude both systems have margin of improvements mostly due to the low number of epochs and size of the training set, however this project can be a good starting point for further studies.

	TRAINING (5k)			VALIDATION		
	Recall	**Precision**	**f1**	**Recall**	**Precision**	**f1**
Rouge-1	0.14	0.24	0.17	0.0004	0.0007	0.0004
Rouge-2	0.07	0.10	0.08	0.00002	0.00004	0.00003
Rouge-L	0.14	0.23	0.17	0.0003	0.0007	0.0004

Table 3: Abstractive performances on training (5k) and validation set

Abstract	Prediction
Syrian official: Obama climbed to the top of the tree, "doesnt́ know how to get down" Obama sends a letter to the heads of the House and Senate . Obama to seek congressional approval on military action against Syria . Aim is to determine whether CW were used, not by whom, says U.N. spokesman .	[START] Syrian official: Obama climbed to the top of the tree, "doesnt́ know how to get down" Obama sends a letter to the heads of the House and Senate . Obama to seek congressional approval on military action against Syria . Aim is to determine whether CW were used, not by whom, says U.N. spokesman . [STOP]

Table 4: Example of prediction after training on one sample

References

[abs] Dataset card for cnn dailymail dataset. "https://huggingface.co/datasets/cnn_dailymail".

[ext] Extractive text summarization on cnn / daily mail. "https://paperswithcode.com/sota/extractive-document-summarization-on-cnn".

[Gehring et al.2017] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

[Nallapati et al.2016] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2016. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *CoRR*, abs/1611.04230.

[pro] Process-data-of-cnn-dailymail. "https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail".

[Raju and Allarpu2017] T Sri Rama Raju and Bhargav Allarpu. 2017. Text summarization using sentence scoring method. *International Research Journal of Engineering and Technology (IRJET)*, 4(04).

[Robertson] Sean Robertson. Nlp from scratch: Translation with a sequence to sequence network and attention. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.

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

[Sutskever et al.2014] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.