NATURAL LANGUAGE PROCESSING: SUMMARIZATION

Casper Andersen, Jacob Johansen, Emilie Trouillard

DTU Compute

ABSTRACT

Index Terms— Deep learning, RNN, NLP, Summarization, Attention, Pointers, Sequence-to-sequence learning.

1. INTRODUCTION

The goal of summarization is to generate a smaller text that contains the main information of an original text and is grammatically correct. Therefore summarization is well fitted for sequence-to-sequence learning. This can seem not too complicated at first sight, but it raises some algorithmic issues.
There are two basic approaches to summarization: extractive and abstractive [1]. The first is simply copying pieces of the source text to build the summary whereas the second is based on reformulation and generates new text. The ideal model uses both methods: abstractive summarization to generate a coherent summary, and extractive summarization to retrieve special words such as names, dates, numbers etc. Word extraction is the less complex part of this task, and as been used for many years in summarization models. However, as we simply copy text from the source, there is a clear limit to how well this method can perform. Therefore people have begun looking into abstratice models, or combinations of abstractive and extractive models, in order to create better summarization models.
The switch from simple extractive model to more advanced models such as neural networks have presented a few challenges; first of all, we need a way to represent the text numerically in order to feed is as an input to a mathematical model. A widely used choice for creating this representation is the word2vec model [2]. This word representation maps words into a high dimensional vector space and produces coordinates such that two words that are related, semantically or grammatically, are close in the vector space [3]. This representation will make paraphrasing possible when building the abstractive part of the summary. The word2vec representation has to be trained as a model on its own to be able to represent the similarities that were presented above. We will however not spend time on this task and use a pre-trained word2vec structure instead, called GloVe. It was introduced by Pennington et. al. [4] and is based on a dictionary $\mathcal{V}$ of around 250k words.

Another question that is encountered when building summarization models is that the length of the output is not known a priori. This is why a simple deep neural network is not able to solve this problem. Some strategies have been implemented to circumvent this issue [5], for instance by using Long Short-Term Memory (LSTM) cells, in an encoder - decoder setup, based on recurrent neural networks (RNNs).
We will build a model following that encoder-decoder structure, but a bit more sophisticated. Indeed, it will be using an attention mechanism, as well as coverage and a pointer, in order to prevent the most common issues that are encountered by summarization networks.

2. RELATED WORK

2.1. Common Structure of the Summarization Models

2.1.1. Sequence-to-sequence Models

Similarly to the Neural Machine Translation, the common strategy for building summarizing models is to first capture the whole input and then generate the summary. That is done by using an encoder decoder structure.
For the summarization models in particular, this encoder decoder relies on sequence-to-sequence learning. It addresses the issue that the input and the output of the model are sequences of unknown length, which is makes it incompatible with traditional Deep Neural Networks (DNN). Sutskever et.al.[5] built an encoder-decoder structure that is able to generate non fixed-length sequences of words. The strategy that is presented in this paper is to use Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) cells. Furthermore, some *end-of-statement* (<EOS>) tokens are appended to the input texts. The idea is that the decoder keeps generating words until it generates this <EOS> and then it stops. Fig. 1 describes the structure of that encoder-decoder structure with <EOS> in a simple way.

2.1.2. Loss Function and Teacher Forcing

It is not obvious how to define a loss function for training the model, considering the fact that there is not only one good summary for a given text. However, we do have labeled data, which means that for each text from the training set we have

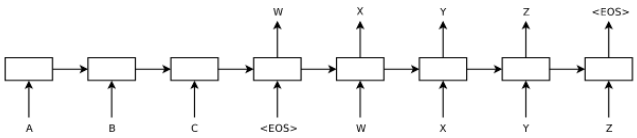

Fig. 1. The model reads an input sentence ABC and produces WXYZ as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier. [5]

a reference summary. The idea is to use the word-by-word generation of the output to compute its loss. At each step, the output of the decoder is a softmax vector of all words of the dictionary, i.e. a vector of probabilities for each word. The loss function for timestep t is then the negative log-likelihood of the target word w_t^* according to the model:

$$\text{loss}_t = -\log P(w_t^*) \tag{1}$$

After each step, its loss is computed according to the above equation, and the t^{th} word of the generated summary is updated with the correct word from the reference. It means that at each step, the previous correct word is fed as input. By doing this, we avoid that the model gets stuck in a wrong direction. It is called teacher forcing.
The loss for the whole sequence generation is then the sum:

$$\text{loss} = \frac{1}{T}\sum_{t=0}^{T}\text{loss}_t \tag{2}$$

2.2. Attention Based Models

The main part of the abstractive summarization is made possible by the attention mechanism. [6] It occurs in the hidden states of the model, and it gives a probability distribution over the words of the input text, changing at each generation step. All the words that have high attention will be considered as a whole in order to output a relevant and precise word, that can summarize them all for the next step. In that way the networks is able to get away from the words that are present in the input text, and looks into the whole dictionary $\mathcal{V}$ to find the best one.

2.3. Coverage Models

Coverage is a measure of the amount of a sequence, that has been summarized. Incorporating coverage in a summarization model helps ensure that all parts of the sequence are considered, while preventing summarizing the same part multiple times. In Neural networks it is customary to implement soft coverage, that is the model is encouraged to pay attention to the entire sequence, without forcing the model to summarize unimportant parts of the sequence.
Coverage is encoded as a vector $\mathcal{C}_{i,j}$, encoding how attention has changed, for the word at position j, h_j, until time i, $\dim(\mathcal{C}_{i,j}) = d$, with d being the length of the sequence. Several ways of constructing and updating the coverage vector exists. A recurrent neural network-based approach may be used for updating the coverage at each time-step:

$$\mathcal{C}_{i,j} = f(C_{i-1,j}, h_j, \alpha_{i,j}, s_{i-1}) \tag{3}$$

where f is a non-linear activation function or a gating function, s_{i-1} is the previous state of the decoder,$\alpha_{i,j}$ is the alignment at time i. A simpler approach, is to at each decoder time-step set the coverage vector equal to the sum of all previous attention distributions:

$$\mathcal{C}^t = \sum_{k=0}^{t-1} a^k \tag{4}$$

the coverage vector is then used as an additional input to the attention model, allowing the model to select unattended parts of the sequence to focus on.

3. DATA SET

We use the *CNN/Daily Mail* dataset [7]. The dataset consists of online news articles and their summaries. The articles contains 781 words on average and the summaries contains 56 words, or 3.75 sentences on average. The dataset comes in two versions; an anonymized and a non-anonymized versions, where the anonmized version has been preprocessed such that each of the name entities in the dataset has been replaced with a unique identifier, e.g. *The United States* and all it's abbreviations (*US*, *USA*, etc.), might be replaced by @entity1. In our implementation we operate on the non-anonymized dataset. This is preferable for a real world implementation, since replacing a name entity and all its abbreviations with a token, is a non trivial task to automate.

4. METHODS

In this section we present three different ideas which we apply to a sequence-to-sequence network in order to obtain better summarization. First we introduce the concept of *attention* and implement a model with attention, similar to that proposed by Napalli et. al. [8]. This model will serve as our initial model, when we compare the added effect of our other two ideas. Next we will apply *pointer-generation* to our model, and lastly we will also apply *coverage*.
The code for the different models can be found at `https://github.com/EmilieTrouillard/DeepLearning_summarization`

4.1. Attention Based Models

Out baserline model will be a sequence-to-sequence model with attention similar to that of Nallapati et al. [8]. The attention is calculated as [9],

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{attention}) \quad (5)$$

$$a^t = \textit{softmax}(e^t), \quad (6)$$

where W_h, W_s and $b_{attention}$ are the trainable weights and bias respectively. We can view the attention distribution, a^t, as a probability distribution over the source text, that tells the decoder where to look, in order produce the next word. We use the attention distribution and the hidden states to calculate the *context vector*, h_t^*,

$$h_t^* = \sum_i a_i^t h_i, \quad (7)$$

where a^t is the attention distribution at time t and h is the hidden state of the network. The context vector can be seen as a fixed size representation of what we have read from the source at this timestep t. We concatenate the context vector the with decoder state s_t and feed the concatenated vector through two linear layers in order to produce our vocabulary distribution P_{vocab}, just as in Abigail et al. [1],

$$P_{vocab} = \text{softmax}(V'(V[s_t, h_t^*] + b) + b'), \quad (8)$$

where V' and V are the weights and b and b' are the biases. P_{vocab} then becomes a probability distribution over all words in the dictionary $\mathcal{V}$, from which we predict words w,

$$P(w) = P_{vocab}(w) \quad (9)$$

We can then calculate the loss for each timstep as well as the global loss following equations 1 and 2.

4.2. Pointer-Generation

An issue of our model, is that it has problems properly including words that are in the input text but not in the overall dictionary $\mathcal{V}$ or even just rare words. It is likely to be the case for names, numbers, dates, e.g. the score '2-0'. Abigail et. al. [1] proposed adding pointer generation to the attention based sequence-to-sequence model to overcome this problem. The idea is that we allow the network to sometimes copy words directly from the input text, instead of only having the option to generate words from the vocabulary. This changes our model in two ways; first of all we are now able to generate out of vocabulary words to our summary by directly copying them from the source text and secondly we are able to copy words that are in the vocabulary but have low probability, by simply putting enough weight on our pointer.

In our pointer generator model we calculate the attention distribution a_t and the hidden state h_t^* as in (6) and (7) respectively. Furthermore we calculate the generation probability, $p_{gen} \in [0, 1]$ for timestep t, from the context vector, h_t^*, the decoder state, s_t, and the decoder input, x_t, as,

$$p_{gen} = \sigma(w_{h^*}^T h^* + w_s^T s_t + w_x^T x_t + b_{pointer}). \quad (10)$$

Here the matrices $w_{h^*}^T, w_s^T, w_x^T$ and the scalar $b_{pointer}$ are the trainable weights and bias respectively and σ is the sigmoid function.

Our network where we have included pointer generation can be seen in Figure 2, it is worth noticing that our final distribution now include the word '2-0' which is not in the vocabulary. Adding pointer generation to our network addresses the problem of having to summarize words with a low vocabulary probability, or words that are not in the vocabulary, e.g. names.

fix figure

4.3. Coverage

A problem we also saw in our baseline model, was the repetition. This might occur in obvious ways, where words are repeated over and over in a nonsense manner, such as 'Germany beat Germany beat Germany...', or a more subtle way where we repeat the same sentences several times throughout the summary. This can be due to the fact that at each step, the model takes the previous word as input, along with the original text. So if we happen to generate twice the same word, the model might end up in a loop of generating the same sequence of words over and over. We therefore add coverage to out model, as described bu Tu et al. [10]. We introduce a coverage vector c^t, which is the sum of attention distribution of all previous decoder timesteps,

$$c^t = \sum_{t'=0}^{t-1} a^{t'}. \quad (11)$$

The coverage vector therefore becomes a sum of the attention distributions, representing how much coverage each word has received at a given time step. We note that c^0 is a zero vector, since no words have received coverage at timestep zero. We use the coverage vector as an extra input to the attention calculations, changing equation (5) to,

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{attention}), \quad (12)$$

where w_c is a trainable weight vector with same length as v. This means that when we calculate attention for a timestep, t, we take into account the previous attention distributions. Abigail et al. suggest modifying the los function as well, such that we punish the network for attentding the same parts of the text. We define this coverage loss as,

$$\text{covloss}_t = \sum_i \min(a_i^t, c_i^t). \quad (13)$$

We should note that the loss function is bounded from above, since $\sum_i a_i^t \leq 1$. The loss function ensures that the network

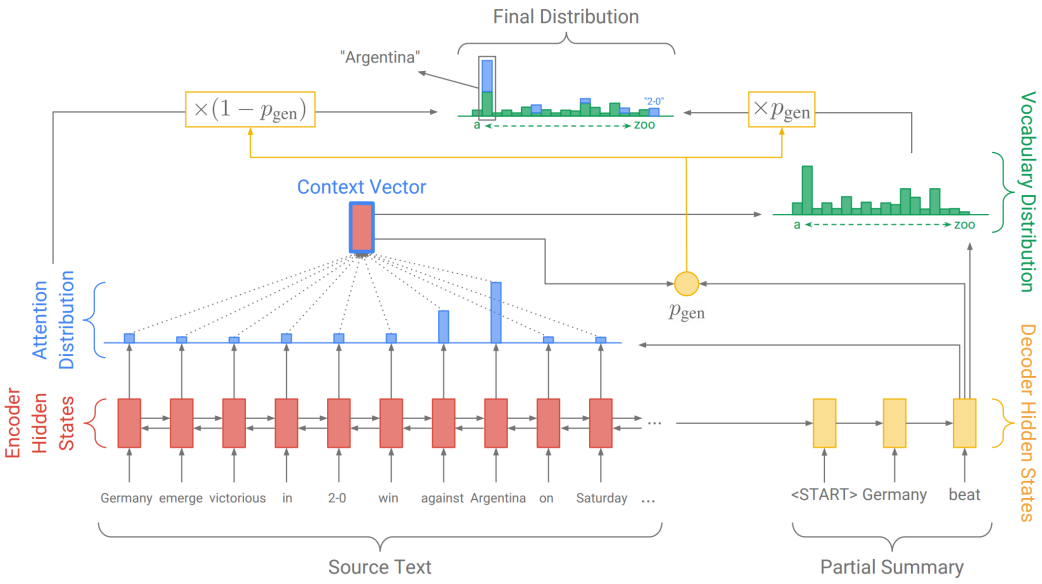

Fig. 2. Our improved model which includes pointer generation. Figure from [1]

is punished for giving attention to the same parts of the text over and over again. In order to ensure that our coverage penalization has the right amount of influence throughout the training, we reweight it by some parameter, λ, this means that our composite loss function for a network with coverage becomes,

$$\text{loss}_t = -\log(P(w_t^*)) + \lambda \sum_i \min(a_i^t, c_i^t). \qquad (14)$$

5. EXPERIMENTAL SETUP

6. RESULTS

7. DISCUSSION

8. CONCLUSION

9. REFERENCES

[1] Abigail See, Peter J. Liu, and Christopher D. Manning, "Get to the point: Summarization with pointer-generator networks," *CoRR*, vol. abs/1704.04368, 2017.

[2] C. McCormick, "Word2vec tutorial - the skip-gram model," 2016, April 19.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

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

[6] Alexander M. Rush, Sumit Chopra, and Jason Weston, "A neural attention model for abstractive sentence summarization," *CoRR*, vol. abs/1509.00685, 2015.

[7] "CNN/Daily Mail Summarization Dataset kernel description," `https://drive.google.com/file/d/0BzQ6rtO2VN95a0c3TlZCWkl3aU0/view`, Accessed: 2018-10-28.

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

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[10] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li, "Coverage-based neural machine translation," *CoRR*, vol. abs/1601.04811, 2016.