Analysis of Word Dependency Relations and Subword Models in Abstractive Text Summarization

Ahmet Beka Özkan
Computer Engineering
Boğaziçi University
Istanbul, Turkey
bekaozkan@gmail.com

Tunga Güngör
Computer Engineering
Boğaziçi University
Istanbul, Turkey
gungort@boun.edu.tr

***Abstract*—Abstractive text summarization is an important task in natural language processing. As many texts are becoming available in the digital world at very high speeds, people begin to need automated systems to summarize such bulk data in a condensed form that only holds the necessary information. With recent advances in deep learning techniques, abstractive text summarization has gained even more attention. Attention-based sequence-to-sequence models are adapted for this task and achieved state-of-the-art results. Additional mechanisms like pointer/generator and coverage have become the standard mechanisms for abstractive summarization. On top of these common approaches, we observe the effects of integrating word dependency relations and using subword models. We show that word dependency relations increase the performance. We also present that subword usage is another viable option to be included for models as well.**

***Index Terms*—text summarization, sequence-to-sequence models, pointer/generator, coverage, word dependency relations, subword models**

I. Introduction

Summarization is the task of acquiring a condensed text considering the main information of an original longer text. There are two different methods for summarization, namely *extractive* and *abstractive*. Extractive summarization selects actual words, phrases or sentences from the input and uses them as the summary. Abstractive summarization, on the other hand, aims to grasp the semantic information of a text and tries to construct a natural, reformed, novel summary that might consist of new words, phrases or sentences, which is similar to human-written summaries.

In this digital age of overloaded information, the need for automatic summarization systems has naturally arisen. Because it is relatively easier than abstractive summarization, researchers have proposed many ways for automatic extractive summarization. However, abstractive systems begin to achieve comparable results with the adaptations of recent deep learning techniques. Because of its innovative potential and the use of external knowledge, abstractive summarization has the potential of producing high-quality summaries.

Semantic information of words is very important in natural language understanding. In order to make the representations of words better, we can also incorporate syntactic information. We saw that very few models were proposed integrating word dependency relations in abstractive summarization. Their effectiveness on such models is not particularly analyzed. In this paper, we integrated word dependency relations and analyzed their effects on abstractive summarization models as a contribution.

Recent deep learning models on Natural Language Processing (NLP) use subword entities. These models have achieved state-of-the-art results and have quickly begun to be used frequently. They can successfully capture the morphological structures of words such as base forms, prefixes, suffixes, etc., which helps natural language understanding and generation. However, there are not many studies using subwords in recurrent neural network (RNN) based models in abstractive summarization. In this paper, we used various subword models and analyze their uses as another contribution.

II. Related Work

Prior to deep learning, abstractive summarization was a big challenge for researchers. Because of the absence of the current natural language understanding and natural language generation techniques, researchers tried to come up with statistical summarization models and graph-based approaches. After the advances of deep learning techniques, abstractive summarization has gained more focus as it becomes an interesting challenge to generate words and summaries from scratch.

Rush et al. [1] designed a fully data-driven model. They used a convolutional network to encode the source sentences and an attentional feed-forward neural network to generate the resulting summaries. They also built the Gigaword dataset for summarization purposes, which is widely used in the literature.

Nallapati et al. [2] proposed an attentional encoder-decoder RNN that is very much similar to the standard machine translation models (Bahdanau et al. [3]). They introduced a two-level hierarchical attention mechanism for both word and sentence levels which provides the model to focus on the important parts of the document more effectively. In addition, they constructed the CNN/Daily Mail dataset, which is also very widely used.

Chopra et al. [4] used almost the same model as Rush et al. Instead of using a feed-forward neural network for generation,

978-1-6654-3405-8/21/$31.00 ©2021 IEEE

they used recurrent neural networks, specifically LSTM and Elman RNN architecture.

Paulus et al. [5] introduced a new intra-decoder attention mechanism, which provides more information flow about the previously generated summary words and the prevention of repeated phrases. The model also has a mixed training objective function that combines both the classical maximum-likelihood cross-entropy loss and policy learning to maximize ROUGE scores directly.

See et al. [6] proposed the pointer/generator and a coverage mechanism. The former is a very good way to solve out-of-vocabulary (OOV) token problems in many abstractive summarization systems since OOV words are generally very important for resulting summaries. The latter is quite useful for not attending the same parts of the input sentences, which means avoiding repetition over the input parts.

Song et al. [7] used the dependency parse tree of the sentences for the pointer-generator mechanism to consider the syntactic constructions better. After obtaining some features from the dependency tree, they proposed using two parallel attention for both words (semantic) and dependency features (structural).

Çelikyılmaz et al. [8] proposed deep communicating agents for abstractive summarization. The encoding task is divided into subsections and handled by multiple agents communicating with each other. Their decoder uses a hierarchical attention mechanism over the agents and the source words.

III. Model

We used two-layer bidirectional RNNs as the encoder that reads the input word embedding vectors $(x_1, x_2, ..., x_n)$ and transforms them into the hidden states $(h_1, h_2, ..., h_n)$, where n is the length of input text. The encoder can be any type of RNN, but typically LSTMs or GRUs are used, and the hidden states obtained from forward $\overrightarrow{h_i}$ and backward $\overleftarrow{h_i}$ passes are concatenated.

$$\begin{aligned}\overrightarrow{h_i} &= \mathrm{GRU}(x_i, \overrightarrow{h_{i-1}}) \\ \overleftarrow{h_i} &= \mathrm{GRU}(x_i, \overleftarrow{h_{i+1}}) \\ h_i &= \overrightarrow{h_i} \oplus \overleftarrow{h_i}\end{aligned} \tag{1}$$

for all $i = \{1, 2, ..., n\}$. Note that $\oplus$ is the concatenation operation. Considering the hyper-parameter of hidden state size as d, $h_i \in \mathbb{R}^{2d}$.

The decoder basically predicts an output token w_t step by step. It is a unidirectional LSTM or GRU network.

$$\begin{aligned}p_{\text{vocab}}(w_t) &= p(w_t \mid w_1, w_2, ..., w_{t-1}, c_t) \\ &= \mathrm{softmax}(W_{\text{v}}(s_t \oplus c_t) + b_{\text{v}})\end{aligned} \tag{2}$$

where $c_t \in \mathbb{R}^{2d}$ is the context vector (shown below), s_t is the hidden state of the decoder at the step t, $W_{\text{v}} \in \mathbb{R}^{v \times 3d}$ is the learnable parameter, and $b_{\text{v}} \in \mathbb{R}^{V}$ is the bias term. Note that V is the vocabulary size.

The decoder updates its hidden state using the embedding vector of the previously generated summary token y_{t-1} and the context vector c_t.

$$s_t = \mathrm{GRU}(y_{t-1} \oplus c_t, s_{t-1}) \tag{3}$$

The context vector c_t is computed with the help of the attention mechanism.

$$\begin{aligned}\alpha_{ti} &= \frac{\exp(e_{ti})}{\sum_{j=1}^{n} \exp(e_{tj})} \\ c_t &= \sum_{i=1}^{n} \alpha_{ti} h_i\end{aligned} \tag{4}$$

The values e_{ti} are computed using (5).

$$e_{ti} = V_a^T \tanh(W_a(s_{t-1} \oplus h_i) + b_a) \tag{5}$$

where $W_a \in \mathbb{R}^{2d \times 2d}$ and $V_a \in \mathbb{R}^{2d}$ are the learnable parameters, and $b_a \in \mathbb{R}^{2d}$ is the bias term.

As usual, negative log-likelihood is used as the loss function.

$$\begin{aligned}\text{loss}_t &= -\log p_{\text{vocab}}(w_t^*) \\ \text{loss} &= \frac{1}{m} \sum_{t=1}^{m} \text{loss}_t\end{aligned} \tag{6}$$

where w_t^* is the predicted word at the decoding step t, and m is the length of the summary.

A. Pointer/Generator Mechanism

The typical usage of the model described above does not have the potential of generating OOV tokens in their original forms. It can only generate a special "OOV" tag from the vocabulary, which represents all of the OOV tokens. However, for a summary to have "OOV" tags are useless because their corresponding original forms generally represent the key information in the article. *Pointer/generator mechanism* (See et al. [6]) is a way of generating OOV tokens in their original forms. It is basically a binary switch that guides the model to either generate a new token from the vocabulary or copy a token from the input text. The probability of the switch is estimated by the value p_{gen} at each decoding step t.

$$p_{\text{gen}} = \sigma(W_g(s_t \oplus c_t \oplus y_{t-1}) + b_g) \tag{7}$$

where $W_g \in \mathbb{R}^{1 \times (3d+E)}$ is the learnable parameter, and $b_g \in \mathbb{R}$ is the bias term. Note that E is the word embedding vector size, and it can be set as a hyper-parameter.

The model generates a token w_t from the vocabulary if the value p_{gen} becomes 1. Conversely, it points to a particular token in the input text if the value becomes 0. When the model opts to point, it copies the token that is being pointed to into the potential summary. The pointing is done with the use of the original attention distribution which is a probability distribution over all of the input tokens regardless of the OOV tokens.

Notice that the probability of generating a particular token is p_{vocab} and can be computed using (2) previously. Now, we

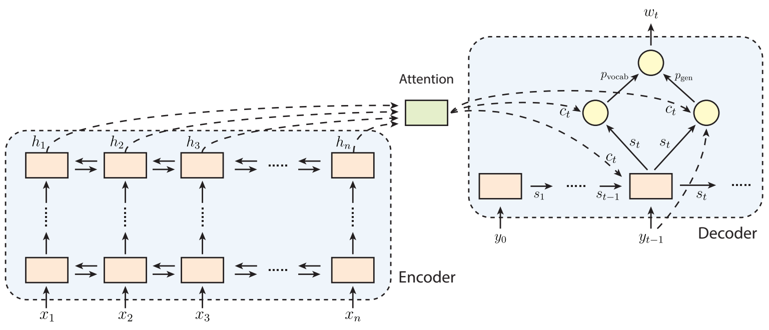

Fig. 1. Sequence-to-Sequence Encoder-Decoder Model with Pointer/Generator Mechanism.

change the probability of producing a particular token to (8) in order to formulate the whole pointer/generator mechanism.

$$p(w_t) = p_{\text{gen}} p_{\text{vocab}}(w_t) + (1 - p_{\text{gen}}) \sum_{i:w_i = w_t} \alpha_{ti} \qquad (8)$$

Fig. 1 graphically represents the model after the addition of the pointer/generator mechanism.

B. Coverage Mechanism

Oftentimes, abstractive summarization models output the same tokens or phrases subsequently. For inputs that are longer articles, it can even generate the same sentence more than once. This obviously is not suitable for summarization. The problem occurs because the attention mechanism actually ignores the past distributions. There is simply no need to focus on the same parts. For this purpose, a new coverage vector is defined (See et al. [6]).

$$\text{cov}_t = \sum_{t'=0}^{t-1} \alpha_{t'i} \qquad (9)$$

We integrate this coverage vector into the attention mechanism, specifically in (5).

$$e_{ti} = V_a^T \tanh(W_a(s_{t-1} \oplus h_i \oplus \text{cov}_t) + b_a) \qquad (10)$$

As See et al. [6] imply, this modification alone does not produce successful results, as the model might opt to ignore this term while training. Therefore, a new loss term is defined as the following.

$$\text{loss}_{\text{cov}_t} = \sum_{i=1}^{m} \min(\alpha_{ti}, \text{cov}_{ti}) \qquad (11)$$

The original loss computation in (6) is also modified with this term.

$$\text{loss}_t = -\log p(w_t^*) + \lambda \text{loss}_{\text{cov}_t} \qquad (12)$$

where λ is a hyper-parameter, and it is typically set as 1 [6].

The coverage loss is actually used as a regularization term. It contributes to the overall loss more if both the coverage vector and the attention distribution for a particularly generated word at any decoding step become high. It consequently forces the model to cover all of the tokens in the input text by forbidding it to attend the same parts. This effectively makes the resulting summaries consider the overall semantic of input text better, and therefore, not miss the important facts and avoid repetitions.

C. Word Dependency Features

Word embeddings are widely used in deep learning. A properly trained word embedding matrix captures the semantic relations successfully. Nevertheless, it does not capture the syntactic features. We can help the models further by integrating syntactic features of words in sentences. We used dependency parsing to obtain a dependency parse tree for each sentence in the datasets. By using the properties of the tree and the results of the parsing, we obtained five syntactic features:

- Part-of-speech tag
- Label of the incoming edge
- Token position in the sentence
- Relative token position in the sentence
- The depth in the parse tree

The number of different part-of-speech tags and the labels of the incoming edges are fixed, which means that it is easy to categorize them. We categorized the relative position feature, which is a real number between 0 and 1, into 10 classes. The remaining features are simply non-negative integers.

After embedding these features for each token in the input, we concatenated the resulting embedding vectors to get an overall structural embedding vector f_i. This vector can be integrated into the model in two different locations. First, the word embedding vector e_i and the structural embedding vector f_i can be concatenated and used as the input of the encoder.

$$e_i = e_i \oplus f_i \qquad (13)$$

Another possibility to integrate is the output of the encoder. This means that the hidden states of the encoder h_i and the structural embedding vector f_i can be concatenated. The attention mechanism and decoder can use these new states as if they are the original outputs of the encoder.

$$h_i = h_i \oplus f_i \qquad (14)$$

D. Subword Usage

Recently, subwords are very widely used in NLP-related deep learning models. With their contributions, many of the new models achieve state-of-the-art results regarding their specific problems. We have found that not many studies in abstractive summarization using RNNs utilized subwords in their models let alone analyzing their effect.

We used byte-pair encoding [9], WordPiece [10] and unigram language model [11] as the subword models. Each of them is trained separately and the resulting subwords are integrated into the models.

TABLE I
THE ROUGE SCORES OF THE MODELS WITH DIFFERENT ADDITIONAL OVER THE GIGAWORD TEST SETS

Gigaword Original								
Model	**R-1**	**R-2**	**R-L**		**Model**	**R-1**	**R-2**	**R-L**
Baseline	29.889	12.402	27.889		Dep Baseline	30.172	12.58	28.019
PG	31.169	13.333	29.142		Dep+PG	31.711	13.403	29.571
COV	21.645	6.718	20.211		Dep+COV	20.468	6.112	19.782
PG+COV	30.494	13.31	28.758		Dep+PG+COV	30.986	13.348	28.998
Gigaword Custom								
Model	**R-1**	**R-2**	**R-L**		**Model**	**R-1**	**R-2**	**R-L**
Baseline	39.019	17.003	36.708		Dep Baseline	39.274	17.513	37.207
PG	41.007	18.264	38.511		Dep+PG	41.747	19.941	39.004
COV	27.962	9.022	26.214		Dep+COV	26.325	9.007	26.179
PG+COV	39.322	17.792	37.217		Dep+PG+COV	39.937	18.48	37.717

TABLE II
THE ROUGE SCORES OF THE MODELS WITH DIFFERENT MECHANISMS OVER THE CNN/DAILY MAIL TEST SET

CNN/Daily Mail								
Model	**R-1**	**R-2**	**R-L**		**Model**	**R-1**	**R-2**	**R-L**
Baseline	31.183	11.769	28.429		Dep Baseline	32.047	11.99	28.718
PG	34.914	14.934	32.025		Dep+PG	34.807	14.072	31.198
COV	25.406	9.923	24.101		Dep+COV	25.278	6.942	22.135
PG+COV	33.023	13.886	30.353		Dep+PG+COV	34.038	14.022	30.989

IV. EXPERIMENTS AND DISCUSSION

A. Preliminaries

a) Datasets: We used the same Gigaword dataset provided by Rush et al. [1] with the same preprocessing workflow, which simply consists of the Penn Treebank tokenization, lower-casing and replacing numeric characters with the special # character. We also created a new Gigaword test set since we think the original one does not resemble the training and validation sets well. We randomly picked 100,000 examples from both the training and validation set and removed them from their respective sets. While training, we clipped the input sentences to make them contain 100 tokens maximum, and the output summaries to 50 tokens. We also used the non-anonymized version of the CNN/Daily Mail dataset through the use of the scripts provided by See et al. [6]. We clipped the input articles to 400 and the output summaries to 100 tokens since it contains larger texts. For both of the datasets, we limited the vocabulary size to 50,000 most frequent unique tokens.

b) Word Embeddings: We used pre-trained GloVe vectors as the word embeddings with the dimension of 200 and froze them while training. We used the same parameters for embedding layers of both the encoder and the decoder.

c) Model Architecture: The baseline models consist of 2-layer bidirectional GRU as the encoder and 1-layer unidirectional GRU as the decoder. Both of them have a hidden vector size of 256. The weights and biases for feed-forward layers are initialized by sampling from a uniform distribution with bounds $\pm\sqrt{\frac{1}{\text{input vector size}}}$. The weights on RNNs are initialized from a uniform distribution with bounds $\pm\sqrt{\frac{1}{\text{hidden vector size}}}$.

d) Training: We used a batch size of 16. The optimization algorithm that we used was Adagrad. We set the learning rate to 0.15 and the initial accumulator value to 0 as its hyper-parameters. We also used gradient clipping with the value 2.0 as the threshold. We used beam search for decoding the output summaries in evaluation, and the beam size was set to 4.

e) Word Dependency Relations: For the models using dependency features in Section IV-B, we used `spaCy`[1], which is a library for NLP in Python.

f) Abbreviations Used in Tables: We used abbreviations for referring to various models we constructed for the sake of simplicity in the subsequent tables. *"PG"* uses the pointer/generator mechanism on top of *"Baseline"*, as described in Section III-A. *"COV"* uses the coverage mechanism only on top of *"Baseline"*, as described in Section III-B. *"PG+COV"* uses both of these mechanisms. Some models have corresponding counterparts that use dependency relations. In order to distinguish them, we used the *"Dep"* prefix. Lastly, we presented the results of the models that use subwords in Section IV-D. Each model that uses a different subword model can be distinguished with a prefix, namely *"BPE"*, which stands for byte-pair encoding, *"WordPiece"* for WordPiece, and *"Unigram"* for unigram language model. They alone do not use any additional mechanism. *"BPE+PG+COV"* uses the pointer/generator and coverage mechanisms on top of *"BPE"*. The only difference between *"BPE+PG+COV"* and *"BPE"* is that the former uses LSTMs as RNNs instead of GRUs. The models using other subword algorithms can be distinguished with corresponding prefixes instead of *"BPE"*. Note that for each table, R-1, R-2 and R-L stand for the F_1 scores of ROUGE-1, ROUGE-2 and ROUGE-L scores, respectively [12]. We used the `pyrouge` library[2], which is a Python wrapper for the original ROUGE summarization evaluation package, to obtain these scores.

B. Dependency Features on Abstractive Models

Tables I and II show the ROUGE scores of the models with different mechanisms described in Section III. It is clear that the integration of the pointer/generator mechanism increases the performance. Every model that has this mechanism achieves better results compared to the baseline models. This shows its usefulness in an abstractive summarization task. With its addition to any model, the ability to generate OOV words is gained. For a summarization task, it is vital to include them in the resulting summaries as explained in Section III-A.

On the other hand, the coverage mechanism does not seem to be useful since it decreases the scores drastically. This is actually a true statement but a misleading one. The actual reason for the low scores is due to the nature of coverage. At the starting time of the training from scratch, the model knows absolutely nothing about the nature of the dataset and abstractive summarization task. We penalize the model by introducing the coverage term to the loss. Therefore, it becomes hard to train the model to get lower loss values. This is why the models with the coverage mechanism produce inconsistent results in Tables I and II.

Generally, the integration of dependency features to each model increases almost every ROUGE score. The only exception seems to be the models with the additional coverage

[1]https://spacy.io

[2]https://github.com/bheinzerling/pyrouge

TABLE III
THE ROUGE SCORES OF "INPUT" AND "HIDDEN" MODELS OVER THE GIGAWORD TEST SETS

	Gigaword Original			Gigaword Custom		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
Input	30.986	13.348	28.998	39.937	18.48	37.717
Hidden	30.19	12.653	28.345	39.015	17.987	37.061

TABLE IV
THE ROUGE SCORES OF "INPUT" AND "HIDDEN" MODELS OVER THE CNN/DAILY MAIL TEST SET

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
Input	34.038	14.022	30.989
Hidden	33.886	13.35	30.334

mechanism only, namely *"Dep+COV"*. The other models generally have increased ROUGE scores by around 0.5-1 in Tables I and II. The improvements of the scores show the benefit of integrating dependency features into abstractive summarization models.

C. Integration Position of Word Dependency Features

We also examined the effect of the placement position of dependency features in the models. *"Input"* uses the equation (13), and *"Hidden"* uses (14). The other parts of these models are the same as *"PG+COV"*. The scores of these models can be seen in Tables III and IV.

We can see that *"Input"* achieves better results than *"Hidden"*. It is due to the fact that the encoder uses them along with the word embeddings to generate better hidden states. These hidden states depend on the dependency features in addition to the word embeddings. Note that we used the same approach as in *"Input"* as the baseline for the models including the use of dependency features in all of the tables above.

D. Subword Models on Abstractive Models

Tables V and VI contain the scores of the models that use different subword models. We observe that the scores of *"BPE"*, *"WordPiece"* and *"Unigram"* are very close to the scores of *"PG+COV"*. Considering that these subword-based models do not use any additional mechanism suitable for abstractive summarization, these results are pretty decent. There are two reasons for that. First, these subword-based models do not use the coverage mechanism. As we concluded in Section IV-C, enabling the coverage mechanism from the start of the training decreases the performance, which is how coverage is used in *"PG+COV"*. The second reason is that the subword models naturally handle the OOV tokens as the pointer/generator mechanism does.

TABLE V
THE ROUGE SCORES OF THE MODELS THAT USE SUBWORD MODELS OVER THE GIGAWORD TEST SETS

	Gigaword Original			Gigaword Custom		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
BPE	30.219	13.08	28.647	39.299	17.212	36.918
BPE+PG+COV	32.186	14.003	30.344	41.973	18.579	40.107
WordPiece	30.24	13.017	28.698	39.401	17.211	37.027
WordPiece+PG+COV	32.201	14.014	30.367	41.997	18.591	40.23
Unigram	30.014	13.01	28.498	38.871	17.894	39.241
Unigram+PG+COV	32.584	13.502	30.681	41.698	18.237	41.149

TABLE VI
THE ROUGE SCORES OF THE MODELS THAT USE SUBWORD MODELS OVER THE CNN/DAILY MAIL TEST SET

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
BPE	33.176	13.89	30.483
BPE+PG+COV	36.196	14.371	33.554
WordPiece	33.204	13.895	30.451
WordPiece+PG+COV	36.196	14.372	33.556
Unigram	33.006	13.925	30.203
Unigram+PG+COV	34.963	14.506	32.972

TABLE VII
THE ROUGE SCORES OF VARIOUS MODELS INCLUDING "DEP+PG→COV" OVER THE GIGAWORD TEST SET

	Gigaword		
Model	**R-1**	**R-2**	**R-L**
ABS (Rush et al., 2015)	29.55	11.32	26.42
ABS+ (Rush et al., 2015)	29.76	11.88	26.96
lvt2k-1sent (Nallapati et al., 2016)	32.67	15.59	30.64
RAS-LSTM (Chopra et al., 2016)	32.55	14.70	30.03
RAS-Elman (Chopra et al., 2016)	33.78	15.97	31.15
UniLM (Dong, Yang, Wang, Wei et al., 2019)	38.90	20.05	36.00
ProphetNet (Qi, Yan, Gong, Liu et al., 2020)	39.51	20.42	36.69
Dep+PG→COV	32.95	15.30	30.97

TABLE VIII
THE ROUGE SCORES OF VARIOUS MODELS INCLUDING "DEP+PG→COV" OVER THE CNN/DAILY MAIL TEST SET

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
words-lvt2k-temp-att (Nallapati et al., 2016)	35.46	13.30	32.65
seq-to-seq + attn baseline (See et al., 2017)	30.49	11.17	28.08
pointer-generator (See et al., 2017)	36.44	15.66	33.42
pointer-generator + coverage (See et al., 2017)	39.53	17.28	36.38
UniLM (Dong, Yang, Wang, Wei et al., 2019)	44.17	21.47	41.11
ProphetNet (Qi, Yan, Gong, Liu et al., 2020)	44.20	21.17	41.30
Dep+PG→COV	38.17	16.93	35.88

The most obvious observation that can be seen in Tables V and VI is that the addition of the pointer/generator mechanism significantly improves the performances. The pointer/generator mechanism is used because of its ability to generate OOV tokens. However, the same ability is inherently gained by using subword models. It could be confusing to understand which factor plays a role in this improvement when they both solve the problem of OOV tokens. The pointer/generator mechanism does not only point to an OOV token seen in the input text. It can also decide to point to a token that is already in the vocabulary. This effectively introduces extractiveness to the models. Indeed, *"BPE+PG+COV"*, *"WordPiece+PG+COV"* and *"Unigram+PG+COV"* have gained extractiveness and this, as a result, helps to increase the scores.

Using subword models alone increases the effectiveness of the models as they solve the problem of OOV tokens. With the addition of the pointer/generator mechanism, these models improve their performances even more. We conclude that integrating subwords into abstractive summarization models is beneficial as the related scores increase in our experiments.

E. Proper Usage of Additional Mechanisms

We also built a model that uses LSTMs instead of GRUs and did not freeze the parameters of word embeddings while training. This model uses the pointer/generator mechanism. We trained the model without using the coverage mechanism until it converges to a sufficient point. Then, we enabled coverage and trained the model further for a while as it shows its benefits in this way. The model also uses dependency features. We call this model *"Dep+PG→COV"*. Tables VII and VIII have the scores of several existing models including *"Dep+PG→COV"* for comparison.

With the proper usage of the coverage mechanism, we can see in Tables VII and VIII that *"Dep+PG→COV"* produces very good results compared to *"Dep+PG+COV"* or *"Dep+PG"* in Tables I and II for both Gigaword and CNN/Daily Mail datasets. It surpasses them by more than 2 points in each ROUGE score for the original Gigaword test set. This shows the benefit of the coverage mechanism in abstractive summarization tasks. On the other hand, the improvement is even higher for the CNN/Daily Mail dataset. The reason for the higher contribution comes from the nature of these two datasets. The input texts of the Gigaword dataset are a lot shorter. The longer texts cause the model to generate repeated words or phrases more frequently. Therefore, using the coverage mechanism for the Gigaword dataset is not as vital as using it for the CNN/Daily Mail dataset.

Table VII contains the results of some popular models evaluated using the Gigaword dataset. We can see that our model can achieve better scores compared to the models by Rush et al. [1], Nallapati et al. and [2], Chopra et al. (except their RAS-Elman model) [4]. In Table VIII, the scores of various models can be seen for the CNN/Daily Mail dataset. Our model can perform better than Nallapati et al. [2]. It also surpasses the baseline and the pointer/generator model from See et al. [6]. However, it fails to achieve better scores than their "pointer-generator + coverage" model. Note that we failed to create a baseline model performing close to their baseline model. We think that integrating dependency features on top of their baseline model would achieve better scores than their best model.

UniLM [13] and ProphetNet [14] are relatively new models and differ from the others in Tables VII and VIII. They are based on transformer models whereas the others use RNN-based networks. It can clearly be seen that these transformer-based models achieve significantly higher results. However, we showed that integration of dependency features increases the performance. Utilizing similar transformer-based approaches with dependency usage might surpass these models.

V. Conclusion

In our experiments, we analyzed the effects of the additional mechanisms used in abstractive summarization models. We have found that the pointer/generator mechanism solves the problem of generating OOV tokens. It also brings extractiveness to the models and balances between these two summarization approaches. We also observed the effect of the coverage mechanism. We have found that using it after the parameters of the models converge in training helps the models avoid any repeated words or phrases, and it is very useful for summarizing long input texts. We further included word dependency relations and found that the knowledge of dependency structure makes natural language understanding more efficient and natural language generation better. Furthermore, we also analyzed the use of three subword models. They naturally handle the out-of-vocabulary word issues in summarization tasks. With the additional mechanisms, the performances of the models get even better.

For future work, we think that other training approaches can be adapted for the models that we used. For example, policy gradient techniques in reinforcement learning to directly maximize the ROUGE scores are used, and they produce worthy results [5]. Recently, transformers are used very frequently, and they achieve state-of-the-art results in many NLP problems, including abstractive summarization. Even though we could not achieve the scores of recent studies, using their models as a baseline and integrate our additions into them might produce better scores. Many popular models use subwords internally. However, finding a suitable way to incorporate word dependency features into these models might provide better results.

References

[1] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*. ACL, 2015, pp. 379–389.

[2] R. Nallapati, B. Zhou, C. N. dos Santos, Ç. Gülçehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence rnns and beyond," in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL*. ACL, 2016, pp. 280–290.

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR*, 2015.

[4] S. Chopra, M. Auli, and A. M. Rush, "Abstractive sentence summarization with attentive recurrent neural networks," in *The Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT*. ACL, 2016, pp. 93–98.

[5] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *Computing Research Repository (CoRR)*, vol. abs/1705.04304, 2017.

[6] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. ACL, 2017, pp. 1073–1083.

[7] K. Song, L. Zhao, and F. Liu, "Structure-infused copy mechanisms for abstractive summarization," in *Proceedings of the 27th International Conference on Computational Linguistics, COLING*. ACL, 2018, pp. 1717–1729.

[8] A. Çelikyılmaz, A. Bosselut, X. He, and Y. Choi, "Deep communicating agents for abstractive summarization," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*. ACL, 2018, pp. 1662–1675.

[9] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. ACL, 2016.

[10] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *International Conference on Acoustics, Speech and Signal Processing, ICASSP*. IEEE, 2012.

[11] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. ACL, 2018, pp. 66–75.

[12] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. ACL, 2004, pp. 74–81.

[13] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon, "Unified language model pre-training for natural language understanding and generation," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019, pp. 13 042–13 054.

[14] W. Qi, Y. Yan, Y. Gong, D. Liu, N. Duan, J. Chen, R. Zhang, and M. Zhou, "Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP*. ACL, 2020, pp. 2401–2410.