# Abstractive Text Summarization with Seq2seq Model: Final Report

Jie Zhou, Shihao Liu, Christos Drou

## Abstract

RNN-based approaches have achieved a great success in abstractive text summarization tasks, however, most of them are using a fixed representation for the tokens and neglecting other word-level or sub-word features. Here we propose a novel Seq2seq model using a hybrid embedding of word, part-of-speech (POS) and characters to explore whether POS and character-level features could help improve the performance in text summarization. And we also attempt to implement two different approaches to integrating additional features. Our models can achieve a small improvement evaluated on ROUGE score, among which the best one is the single-encoder model with character-level features. When using character-level features, our model can make a better improvement on the subset where the document has more rare words. However, the ability for reparaphrasing of our models is still limited.

## 1. Introduction

Text summarization is the task of condensing a piece of long document to a shorter one containing the main information in the source article. There are two general approaches for summarization: *Extractive and Abstractive*. In extractive methods, words, phrases and sentences are directly selected from the source text, which therefore are relatively easy to implement and can often produce grammatically correct results. However, some other advanced capacities such as reparaphrasing, generalization are impossible in the naive extractive framework, but can be achieved by abstractive ways.

Recently, recurrent neural networks (RNN), a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence, have gathered a lot of popularity in sequential modelling. Due to their effectiveness in modelling long distance dependency and flexibility to handle sequence in arbitrary length, besides text summarization, RNN models have also been successfully applied to a variety of other natural language processing (NLP) tasks, like machine translation(Bahdanau et al., 2014), (Wu et al., 2016), language modelling (Mikolov et al., 2010), etc. While in the field of text summarization, RNN-based methods can not only achieve a better performance in terms of ROUGE (Lin, 2004), but also presents great potential for incorporating real-world knowledge. And most of them (Rush et al., 2015), (Nallapati et al., 2016) are a sequence-to-sequence (seq2seq, or say encoder-decoder) architecture, where a RNN layer acts as "encoder" which processes the input sequence and returns its own internal state, and another RNN acts as "decoder" trained to predict the next token of the target sequence, given previous tokens of the target sequence and the source sequence.

Though these seq2seq models are promising, several shortcomings are still expected to overcome: 1) inaccurate reproduction of the salient information of source documents. 2) terrible handling of out-of-vocabulary (OOV) words. 3) suffering from self-repetitions and unnatural summaries. A variety of extensions of the seq2seq architecture were proposed in order to fix these problems, most of them are using sentence-level or even more global features. (Tan et al., 2017) proposed a graph-based attention framework to discover the salient information of a document. (Hsu et al., 2018) leveraged the sentence-level salient information from an extractive model to modulate the word-level dynamic attention from the abstractive model. And (Chung et al., 2018) took the information of whole document into consideration rather than just relying on the source sentence at each decoding step.

However, almost all of the previous methods use a fixed representation for the tokens and neglecting other word-level or sub-word features. These methods might not be so good at word sense disambiguation and also not generalize well to unseen or rare word forms, which indirectly diminishes the performance of text summarization.

In order to endow the model with the ability of word sense disambiguation and interpreting unseen or rare word, we enrich the representation of words with POS tagging and character-level features. Attaching POS to each token can directly introduce contextual information to the model, it might help to separate different senses from the same word form, and handle potential grammar issues. While by encoding the information of characters, we can enable the model to detect more fine-grained structure, like morphemes, and even to speculate the meaning of an unseen word given its combination of n-grams.

To our best knowledge, we have not seen any paper combining both these features for the text summarization task. The research question we are going to explore is that, to what extent, and in what aspect, the character-level and POS features can help to improve the performance in text summarization.

## 2. Data set and task

Our task is to build a seq2seq model using both word and sub-word level representation (described in Section 3) for abstractive text summarization. The baseline is a pointer-generator network with coverage mechanism(See et al., 2017).

### 2.1. CNN/Daily Mail dataset

The dataset we are using is CNN/Daily Mail (Hermann et al., 2015) (Nallapati et al., 2016) which includes news articles (39 sentences on average) paired with multi-sentence summaries. Specifically, we use 287,226 training pairs, 13,368 validation pairs and 5,000 test pairs (references), and the data split scheme and preprocessing procedure is following the setting proposed by (See et al., 2017) [1]. The original number of the test pairs in the dataset were 11,490, but we randomly select 5,000 of them in order to reduce the test time of the experiments. We are using the non-anonymized version of dataset.

### 2.2. ROUGE Metric

We will evaluate our models with the standard ROUGE package (Recall-Oriented Understanding for Gisting Evaluation) (Lin, 2004). This package includes measures to automatically determine the quality of a summary by comparing it to summaries created by humans (golden standard). The most common metrics in this package is the ROUGE-N (n-gram co-occurances statistics) and the ROUGE-L(Longest Common Subsequence).

#### 2.2.1. ROUGE-N

ROUGE-N is an n-gram recall between the generated summary and the reference summaries and it is computed as follows:

$$ROUGE-N = \frac{\sum_{S \in References} \sum_{gram_n \in S} count_{match}(gram_n)}{\sum_{S \in References} \sum_{gram_n \in S} count(gram_n)}$$

where $S$ stands for the reference summary, $n$ stands for the length of the n-gram and $count_{match}(gram_n)$ is the maximum number of n-grams that co-occur both in the set of the generated summaries and the set of the reference.

#### 2.2.2. ROUGE-L

The intuition behind ROUGE-L is that the longer is the Longest Common Subsequence between 2 sentences the more similar they are. In summary level we take the union LCS matches between a reference summary sentence $r_i$ and every candidate summary sentence $c_j$. Given a reference summary of $u$ sentences containing a total of $m$ words and a candidate summary of $v$ sentences containing a total number of $n$ words, and assuming that k = $LCS_{\cup}(r_i, C)$ is the union of the longest common subsequences of $r_i$ and the sentences $c_i \in C$, we compute:

$$\text{ROUGE-L} = \frac{2pr}{r+p}$$

where $p = \frac{\sum_{i=1}^{u} k}{n}$ and $r = \frac{\sum_{i=1}^{u} k}{m}$.

#### 2.2.3. Limitation on ROUGE

Practically, ROUGE measures show how much content of the reference summary (golden standard) is captured by the system summary. However, ROUGE does not give a definitive understanding of the performance of the system summaries in comparison to human summaries. For example, it can not capture synonymous concepts; ROUGE-N compares n-gram overlap of words without detecting synonymous terms. The ROUGE scores also expect system summaries to be identical to reference. Furthermore, these metrics can not capture if different subset of content or topics have been correctly covered by the system summary.

## 3. Methodology

In this section we describe the architecture of our models. They are based on the seq2seq model proposed by (See et al., 2017), which combines attention, pointer-generator and coverage mechanism. We also propose two different encoder architectures explore which is the better way to integrate POS and character-level features into networks in this article.

Detailed description about the calculation of each representation would be presented in Section 3.1. Two architectures of encoder would be shown in Section 3.2. The mechanism of decoder would be described in Section 3.3. In Section 3.4, 3.5 and 3.6 we present the attention, pointer-generator and coverage mechanism respectively.

### 3.1. Features

#### 3.1.1. Word-level representation

Let $V_w$ be the vocabulary of the words. We use a dense vector representation for words with dimensionality $d_w$, i.e. word embeddings. We get the word vector $\mathbf{v}_{word}$ by extracting the corresponding row from an embedding matrix $E_{word} \in \mathbb{R}^{|V_w| \times d_w}$, which is a learnable parameter.

#### 3.1.2. POS representation

Let $V_p$ be the set of the POS tags. Similarly to Section 3.1.1, a POS vector $\mathbf{v}_{pos}$ with dimensionality of $d_{pos}$, is obtained from the POS embedding matrix $E_{pos} \in \mathbb{R}^{|V_p| \times d_{pos}}$. In Section 4.2 we will discuss in more details about the tools used for POS tagging.

#### 3.1.3. Character-level representation

Let $V_c$ be the vocabulary of characters, $d_c$ be the dimensionality of character embeddings. Similar to Section 3.1.1, we use a character embedding matrix $E_{char} \in \mathbb{R}^{|V_c| \times d_c}$. Suppose that the word $k \in V_w$ is made up of a sequence of

[1] See the github page https://github.com/abisee/cnn-dailymail

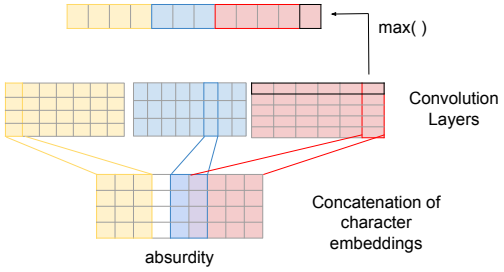


*Figure 1.* Character convolution DEMO for the word "absurdity", using 3 filters with width 2,3 and 4. The diagram is derived from (Kim et al., 2016). Note that the width and number of filters here are only for demonstration, refer Section 4.3 for specific setting in our work.

characters $c_1$, $c_2$, ...,$c_l$, where $l$ is the length of the word $k$. Then the character level representation of $k$ is given by the matrix $C^k \in \mathbb{R}^{d_c \times l}$, where the $j$-th column corresponds to the character embedding for $c_j$.

Following the methodology of (Kim et al., 2016), we apply a 1-d convolution with slide as 1 between $C^k$ and a filter (or kernel) $W \in \mathbb{R}^{d_c \times w}$ of width $w$. Note that kernels are all learnable parameters. Then we add a bias and apply a nonlinearity transformation to obtain the feature map $f^k \in \mathbb{R}^{l-w+1}$. Specifically, the $i$-th element of $f^k$ is given by:

$$f^k[i] = tanh(\langle C^k[*, i : i + w - 1], W\rangle + b)$$

where $C^k[*, i : i + w - 1]$ is the $i$-to-$(i + w - 1)$-th columns of $C^k$ and $\langle A, B\rangle$=Tr($AB^\intercal$) is the Fobenious inner product. Note here is the equation for the "valid" padding in convolution, the dimension of feature map can still be $l$ if using the "same" padding.

Finally we take the max-over-time $v^k = \max_i f^k[i]$ as the feature corresponding to the filter $W$. The idea behind the operation of "max-over-time" is to capture the most important feature, i.e. the one with the maximum value for a given filter. A filter is essentially picking out a character $n$-gram, of which the size of the $n$-gram corresponds to the filter width.

Our CNN uses multiple filters of varying widths to obtain the feature vector for the word $k$. So if we have a total of $n$ filters $W_1, .., W_n$, then $\mathbf{v}^k_{char} = [v^k_1, ..., v^k_n]$ is the final character-level feature vector of word $k$.

### 3.2. Encoder

#### 3.2.1. Single Encoder

The first model uses a similar way to the Feature-rich Encoder proposed by (Nallapati et al., 2016), which simply using concatenation to combine additional features into model. As shown in Figure 2, the architecture is quite similar to the pointer-generator network (See et al., 2017), The only difference is the input of the encoder. In our network, we can concatenate any two or them or even all of them together, which is supposed to greatly enrich the features the model can leverage. More specifically, one can simply enrich the word embeddings with POS and character-level features by feeding the concatenated vector into LSTM cell, i.e. $[\mathbf{v}_{word}; \mathbf{v}_{pos}; \mathbf{v}_{char}]$.

Here the encoder is a single-layer bidirectional LSTM (Hochreiter & Schmidhuber, 1997). Using bidirectional LSTM, practically we do a forward and a backward pass in the input text in order to capture the information from the whole context. Since we use a bidirectional encoder but an one-directional decoder (in Section 3.3), some transformation might be needed to match the dimensionality between the final state of encoder and the initial state of decoder. Assume the final states of encoder in two direction are $\overrightarrow{h_I}$, $\overleftarrow{h_I}$ separately ($I$ is the length of input sequence). We obtain the initial state $s_0$ of decoder by doing a non-linear mapping on the concatenation of two final hidden states $\overrightarrow{h_I}$ and $\overleftarrow{h_I}$:

$$s_0 = relu(W_{mult}[\overrightarrow{h_I}; \overleftarrow{h_I}] + b_{mult}) \quad (1)$$

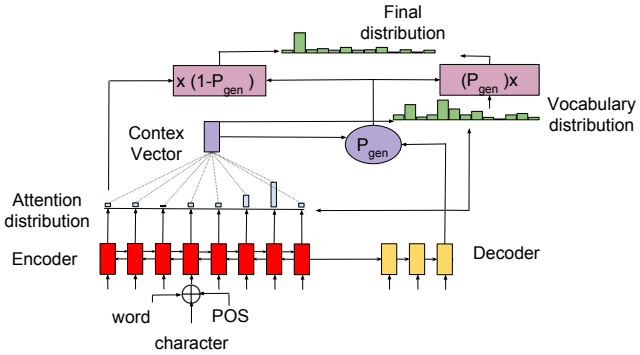


*Figure 2.* Single-encoder method for integrating additional features into network.

#### 3.2.2. Multiple Encoders

Our second model (Section 3.2.2) uses multiple encoders, each of them handling character-level, word-level and POS representation separately. We feed each encoder with the word, POS and character feature vectors respectively (as shown in Figure 3). And then the final hidden states of each encoder, $\overrightarrow{h_I^{word}}$, $\overleftarrow{h_I^{word}}$, $\overrightarrow{h_I^{char}}$, $\overleftarrow{h_I^{char}}$, $\overrightarrow{h_I^{pos}}$ and $\overleftarrow{h_I^{pos}}$ would be combined together using similar method to Section 2. Specifically, we concatenate all the final states in a certain order and apply the same non-linear transformation as Equation 1:

$$s_0 = relu(W_{mult}[\overrightarrow{h_I^{word}}; \overleftarrow{h_I^{word}}; \overrightarrow{h_I^{char}}; \overleftarrow{h_I^{char}}; \overrightarrow{h_I^{pos}}; \overleftarrow{h_i^{pos}}] + b_{mult})$$

One can change the contents in the concatenated vector when using different features. In the second encoder architecture, we will have more free parameters and all the information from the POS and character-level feaures would be passed to the decoder through the above equation.

### 3.3. Decoder

On each step $t$, the decoder (a single layer undirectional LSTM) receives just the word embedding of the previous

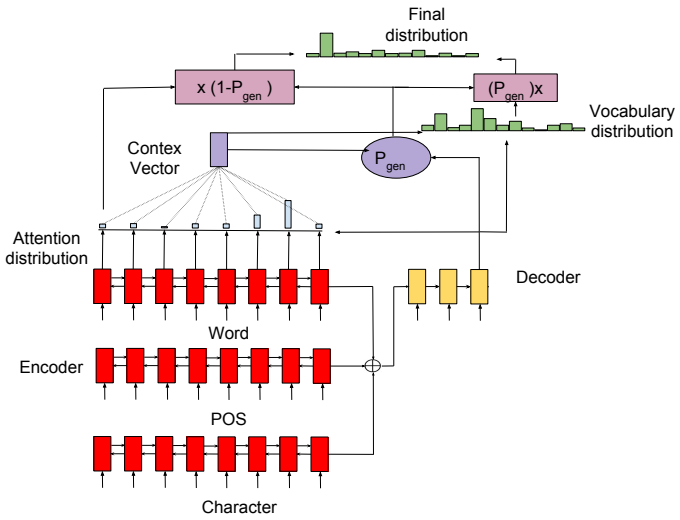


*Figure 3.* Multi-encoder method for integrating additional features into network.

word (at training time this is the previous word of the reference summary, at test time it is the previous word emitted by the decoder) and has a decoder state $s_t$. And it will also receive the final state from encoder as its initial state $s_0$ (see the calculation of $s_0$ in Section 3.2.1 and 3.2.2).

### 3.4. Attention Mechanism

According to (Bahdanau et al., 2014) the attention distribution is calculated as:

$$e_i^t = u^t tanh(W_h h_i + W_s s_t + b_{attn}) \quad (2)$$

$$a^t = softmax(e^t)$$

where $h_i$ is the encoder hidden state at $i$-th token, $u$, $W_h$, $W_s$ and $b_{attn}$ are learnable parameters. Essentially, the attention distribution over the source token are just telling the decoder which word should be attended to when producing the next word. Then the context vector $h_t^*$ is computed as the weighted sum of the encoder hidden states:

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

The context vector is concatenated with the decoder state $s_t$ and fed through two linear mappings to get the vocabulary distribution $P_{vocab}$:

$$P_{vocab} = SoftMax(U'(U[s_t; h_t^*] + b) + b')$$

where $U$, $U'$, $b$ and $b'$ are all learnable parameters. And $P_{vocab}$ is a probability distribution over vocabulary and provides us with our final distribution for predicting words $w$:

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

During training, the loss for timestep $t$ is the negative log likelihood of the target word $w_t^*$:

$$loss_t = -logP(w_t^*)$$

and the overall loss for the given sequence is:

$$loss = \frac{1}{T} \sum_{t=0}^{T} loss_t$$

### 3.5. Pointer-generator

The pointer mechanism allows us both copying word via pointing and generating words from a fixed dictionary. The generation probability $p_{gen} \in [0, 1]$ is computed from the context vector $h_t^*$, the decoder state $s_t$ and the decoder input $x_t$ (we feed word vector without any additional features to the decoder):

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

$p_{gen}$ behaves like a switch to choose between generating a word from the vocabulary by sampling from $P_{vocab}$, or just copying from the input sequence by sampling from the attention distribution $a_t$. We refer the union of the vocabulary and all words appearing in the source document as "extended vocabulary". We obtain the following probability distribution over the "extended vocabulary":

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

### 3.6. Coverage Mechanism

In our model, the coverage mechanism, introduced by (Tu et al., 2016) and applied to abstractive summarization by(See et al., 2017), is used for handling the problem of self-repetition. We use a coverage vector $c_i$ which is the sum of attention distribution for all the previous decoder time steps:

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

Intuitively, $c^t$ is a (unnormalized) distribution over the source words that represents the degree of coverage that those words have received from the attention mechanism so far. The coverage vector is used as extra input to the attention mechanism, changing the equation of attention distribution as below:

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

Therefore, the attention mechanism's current decision (choosing where to attend next) can be influenced by a reminder of its previous decisions. This reminder is all the information summarized in the coverage vector $c^t$.

According to (See et al., 2017), in order to penalize repeatedly attending to the same locations, the coverage loss is defined as:

$$covloss_t = \sum_i min(a_i^t, c_i^t)$$

The idea is that we penalize the overlap between each attention distribution and the coverage so far. In other words, we will prevent words that received high attention in the past,

from having high attention in the present. Thus, we will avoid repeatedly attending to the same locations, and by extension, we will avoid generating repetitive text. We define our final loss function as the coverage loss reweighted by some hyperparameter $\lambda$ plus the original loss function:

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

## 4. Experiments

In this section, we will describe the specific setting for each experiment and present the results.

### 4.1. General Setting-up and Baseline

Each model is trained on a single Nvidia Tesla P100 GPU for around 80,000 iterations (about 40 hours), where only the latest 20,000 iterations would add the coverage loss (see Section 3.6). And we would only present the score without coverage loss in baseline model, and results of all other models with additional features are assumed to be trained through such a two-stage process.

We train our models using Adagrad (Duchi et al., 2011) with an initial accelerator value of 0.1 and gradient clipping with a maximum gradient norm of 2. The learning rate is set as a constant 0.15, and batch size is 16. Based on (See et al., 2017)'s recommendation, the coverage weight $\lambda$ is set as 1. Also, we use a moving average loss to implement early stopping.

The hidden state dimension in LSTM cell of the encoder and decoder are both set 256. And we set vocabulary size as 50,000, which is the top 50,000 in the word frequency table in the training data. We also set the dimension of word vectors as 128, and truncate the articles longer than 400 words. During the decoding stage, a beam search with width of 4 is used in this work. We also limit the number of decoding steps under 100.

Following the above setting, we rerun the experiments implemented by (See et al., 2017), because of the limitation of time and computational resources (they run over 600,000 iterations, whereas we only train 80,000 iterations), we get the result slightly worse than those in their work in the model without coverage loss (ROUGE-1 score 36.07 v.s. 36.44). After adding the coverage loss, the improvement is not so significant as their work either. These re-trained models are referred as "baseline" and "baseline with coverage" in this article (shown in the top panel in Table 1).

### 4.2. Models with POS Feature

We use the Penn Treebank part-of-speech tags (Marcus et al., 1993) in this work, where the total POS category number is 45. We train a *BackoffTagger* [2] using NLTK toolkit (Bird, 2006), which for each token to be tagged, consults *TigramTagger*, *BigramTagger*, *UnigramTagger*, *AffixTagger* and *RegexpTagger* in order. The *TigramTagger* gives the most probable POS tag given two previous tags, while bigram and unigram taggers work similarly given one previous tag and no previous tag respectively. And *AffixTagger* gives the most probable tag given the 3-character-long suffix. And the final *RegexpTagger* uses regular expressions to match words to specific POS Tags. The training data is 3522 sentences of the PennTreebank corpus provided by NLTK, which is only a part of the full Penn Treebank. The average F1 score on its testset(392 sentences) is 92.0%.

All articles in our training, validation and test set would be annotated by this tagger. As mentioned in Section 3.1.2 we also use a 32-d dense vector to represent the POS tags in our system, which would be trained in an end-to-end way.

The POS feature is combined with word level representations in two different ways, i.e. through the single-encoder and the multi-encoder way (see Section 3.2.1 and 3.2.2). Following the general setting in Section 4.1, we run experiments with word + pos using single-encoder and multi-encoder separately, through which we would like to explore how the POS features and how the method of combining feature into network affect the performance for summerization.

### 4.3. Models with Character Features

The number of characters we consider in our work is 67 in total, including 26 lower case English letters, digits 0-9 and other common special symbols as following:

, ; .!? :′ ”/|_@#$% ∧ & ∗ ‘ + − =<> ()[]{}

As shown in Section 3.1.3, here we set the dimensionality $d_c$ of character embeddings as 16. In the stage of convolution, we set the number of filters $n$ as 16, and the width for each 4 filters is 2, 3 ,4 and 5 respectively. Thus the dimension of final character feature vector $\mathbf{v}_{char}$ for each word is still 16.

While during implementation, we add two special token $< w >$ and $< /w >$ to denote the word boundary. And we would truncate words longer than 32 characters, we also pad special tokens for those shorter words to 32 characters long.

Similarly, to test the performance of models using character-level features, We combine the word vectors with them in two different ways and run the experiments following the general setting (in Section 4.1).

### 4.4. Models with both POS and Character Features

We combine both POS and character-level features in the last part of our experiments to see the behavior of our model when having both two kinds of features. Again, we run two experiments using different encoder architectures. Since the multi-encoder model with both POS and character features has the most free parameters, we train the model 80,000 iterations without coverage and additional 20,000 iterations with coverage loss.

[2] see the NLTK POS tagging tutorial through https://www.nltk.org/book/ch05.html

| Categories | Model | Rouge-1 | Rouge-2 | Rouge-L |
|---|---|---|---|---|
| Baseline | pointer-generator | 36.44 | 15.66 | 33.42 |
| | pointer-generator + coverage | 39.53 | 17.28 | 36.38 |
| | pointer-generator(ours) | 36.07 | 15.22 | 32.73 |
| | pointer-generator+coverage(ours) | 36.84 | 15.86 | 33.45 |
| POS | pos-Single | 37.23 | 15.72 | 33.39 |
| | pos-Multi | 37.46 | 16.16 | 34.01 |
| Char | char-Single | **37.84** | **16.47** | **34.44** |
| | char-Multi | 37.11 | 15.93 | 33.75 |
| Char-POS | char-pos-Single | 37.23 | 16.19 | 33.99 |
| | char-pos-Multi | 37.02 | 15.89 | 33.66 |

*Table 1.* ROUGE f1 scores on test data. Top panel is the result presented by (See et al., 2017) and our reproduced baselines.

### 4.5. Results

The results in Table 1 show that our models all outperform the baseline and the baseline with coverage. Because of its ability to reduce repetition, the coverage mechanism could lift ROUGE-1, 2, L score around 0.6 ~ 0.8 on the baseline, and also get a similar improvement on other models with additional features in our experiments (not shown in this article).

The best performance is achieved by the single-encoder model with character-level features, which promotes the score of ROUGE-1 from 36.84 (baseline with coverage) to 37.84, ROUGE-2 from 15.86 to 16.47 and ROUGE-L from 33.45 to 34.44.

Different features as well as different methods of integration can all have different effect on model's performance. In the POS category, Multi-encoder is better than single-encoder method, but when it comes to only introducing character-level features, the situation is totally opposite. However, we also observe that the model with all the word, POS and character features is not as good as the one with only one kind of extra features. The multi-encoder model with both POS and character-level features is even the worst one. Deeper and more detailed analysis into the interaction of these two features would be done in the future.

## 5. Discussion

### 5.1. The Effect of POS Features

From the results in Table 1, we find that the models augmenting POS features, no matter by single- or multi- encoder, outperform the baseline model. To further analyze the performance, we choose to look into the summaries that generated by the models and we figure out that by adding POS features, the model is more capable of capturing the *attributive connectives*. The Table 2 below shows some specific examples of the baseline and models with POS features.

These two series of examples show that while the baseline fails to capture the information of attributive clause, the models with POS features can apparently locate the position of the *antecedent* in the sentence, as well as its *relative pronoun* such as "who", "whom", "that" and "which",

| model | summary |
|---|---|
| reference | ①...Aracely Meza, **who is not the child 's mother,** was arrested on monday. She has been charged with injury to a child by omission ...<br>②...Billy Mitchell was born with apert syndrome, a rare condition **that causes malformations** to the skull, face, hands and feet ... |
| baseline | ①...Aracely Meza has been arrested in the case involving an attempt to resurrect a two-year-old boy at a home operating also as a church ...<br>②...Billy Mitchell, seven, was born with apert syndrome, a rare genetic condition ... |
| POS_models | ①...Aracely Meza, **who is reportedly not the child 's mother,** was arrested on monday and has been charged with injury to a child ...<br>②...Billy Mitchell, seven, was born with apert syndrome, a rare genetic condition **which causes malformations** ... |

*Table 2.* Example summaries from reference, baseline model and POS models. Models with POS features are sensitive to the *antecedent*, thus could keep the attributive clause, i.e. words in **bold** form. See the full article by following these links: example ①, example ②.

so that the model will attain the attributive clause in the sentence. For example, compared to the summary generated by POS models, the baseline's output in example ② misses the clause information "that causes malformations", which is actually the essential information for the readers to understand what kind of "condition" the article is talking about. From the perspective of POS, the word "condition" is a noun (NN) whereas the word "which" or "that" is a wh-determiner (WDT). It is apparent that NN would be possibly followed by WDT in a sentence, hence, the model is good at capturing this kind of pattern. However, the baseline model only relies on the word embeddings including a lot of information of word sense, context and other redundant messages, which means that the model would assign little weight to POS features and would weaken the effect of POS representations among all these information.

### 5.2. The Effect of Character-level Features

The motivation of using character-level embedding is to strengthen the model's capacity of understanding rare or unseen words. Thus in this section, we will discuss the effect of character-level features for documents with rare words in terms of the ROUGE score, then include some example summaries to demonstrate the behavior of our model intuitively.

Here we treat all words ranked before 40,000 in the word

| SUBSET | BASELINE WITH COVERAGE | CHAR-SINGLE | CHAR-MULTI |
|---|---|---|---|
| RARE WORDS >= 20 | 32.06 | 33.53 | 32.85 |
| %IMPROVE | - | **4.59** | **2.46** |
| RARE WORDS <= 5 | 38.75 | 40.11 | 39.13 |
| %IMPROVE | - | 3.51 | 0.98 |
| TOTAL | 36.84 | 37.84 | 37.11 |
| %IMPROVE | - | 2.71 | 0.73 |

*Table 3.* ROUGE-1 f1 score of models using character-level features on different subsets (mentioned in Section 5.2) and the total test data respectively. The improvement of each model is compared with the baseline with coverage.

frequency table of training corpus as "common words", while other words appearing in the test data are defined as "rare words", i.e. words ranked after 40,000 in the frequency table and OOV words. Note that our vocabulary is the top 50,000 words in the word frequency table.

We filter 773 documents with >= 20 rare words and 1071 documents with <= 5 rare words in the test data, and evaluate the ROUGE-1 f1 score on the selected two subsets. As shown in Table 3, the ROUGE-1 score in the subset is much higher in the subset "rare words <=5" than that in the subset "rare words >=20". But we find that improvement compared with baseline made by models leveraging character-level features on the subset "rare words >=20" is higher than that on the whole test set as well as on the subset "rare words <=5", especially on the "Char-Multi" model.

One example summary is shown in Table 4. These 3 models presented here all produce nearly the same summaries, but the models with character features successfully capture the rare word *'hominins'* and match the golden standard which leads to higher ROUGE score.

### 5.3. How Abstractive Can Our Model be?

In this section, we will measure the ability of our model to create new words when it does summarization. As shown in the Table 5, all the models trained in our work do not create many new words on their summaries; the ratio of the number of new words to that of total words ranges from 0.01% to 0.03%, which indicates that our models still are "extractive" most of the time. Additional, the newly created words are also quite limited, most of which are concentrated in the common words in the sports news as following:

"beat", "score", "win" , "show", "say", "tell", "news".

(See et al., 2017) also reported similar phenomenon in their work and the reason might be that some references (golden standard) for this summarization task are even exactly the original sentences (or slightly modified) extracted from the full document.

| model | summary |
|---|---|
| reference | Spanish researchers say climate change impacted human migration. Until 1.4 million years ago it was too cold to inhabit southeast Spain. But then the climate warmed to 13°C (55°F) and became more humid. This enabled **hominins** - our distant ancestors - to move to new regions. |
| baseline | Researchers believe the spread of our distant human ancestors had been halted by colder and drier temperatures ... |
| Char-Single | Researchers believe that the spread of our distant human ancestors, **the hominins**, had been halted by colder and drier temperatures ... |
| Char-Multi | Researchers believe the spread of our distant human ancestors, **the hominins**, had been halted by colder and drier temperatures ... |

*Table 4.* Example summaries. Words in **bold** form are the "rare words" captured by our models.See the full article by following this link.

| CATEGORIES | MODEL | NO. NEW WORDS | NO. TOTAL WORDS |
|---|---|---|---|
| Baseline | baseline | 85 | 290,511 |
| | baseline with coverage | 43 | 285,317 |
| POS | pos-Single | 73 | 281,065 |
| | pos-Multi | 80 | 260,153 |
| Char | char-Single | 53 | 300,642 |
| | char-Multi | 55 | 341,931 |
| Char-POS | char-pos-Single | 78 | 281,065 |
| | char-pos-Multi | 66 | 260,153 |

*Table 5.* The number of new words and total words in the summaries generated by each model over the test data. The total number of words in full articles of test data is 4,169,253, while the average full article length is 833.85 tokens.

## 6. Related work

**Neural Abstractive Summarization.** Inspired by the success of neural machine translation (NMT), many works (Rush et al., 2015), (Nallapati et al., 2016) on abstractive text summarization followed their seq2seq architecture, and achieved a significant performance improvement over traditional methods. But early works could not handle OOV words and also suffered from self-repetition. In order to tackle OOV words problems, a series of pointing/copying mechanisms (See et al., 2017), (Gulcehre et al., 2016) were proposed, which enables the model to copy words from source text. And the self-repetition problem can be addressed by the coverage mechanism (See et al., 2017), intra-temporal and intra-decoder attention mechanisms (Paulus et al., 2017).

Recent progress has been seen in integrating text summa-

rization into the reinforcement learning framework, where the model is viewed as an agent, each decoding step is corresponding to selecting an action according to a policy. (Zhang & Lapata, 2017) proposed a REINFORCE algorithm, and jointly modelled simplicity, grammaticality, and semantic fidelity to the input. (Ranzato et al., 2015) modified REINFORCE algorithm by incorporating the idea of curriculum learning strategy, which could improve convergence and stability of the training. And the self-critic sequence training approach (Paulus et al., 2017) also showed low variance for the gradient estimator.

Due to its advantage in parallel computing, convolutional seq2seq model has attracted wide attention and also been successfully applied in text summarization task (Fan et al., 2018), (Gehring et al., 2017).

**Character-level Features.** Due to their ability of modelling sub-word level structure, character embeddings have been widely used in various tasks in NLP. In the field of Language Modelling for morphological rich languages, (Kim et al., 2016) proposed a character-aware language model by combining the word vector with the output of character-level convolution on each word, and found that the character N-gram representations successfully learned to understand the suffix and prefix of words. In order to handle the OOV words on the informal text, (Liang et al., 2017) used an multi-encoder model and a Highway Network to combine the word and character embedding and achieved a significant performance improvement in relation classification task. While FastText (Bojanowski et al., 2017) entirely abandoned to model an integral word, it represented each single word with a bag of character n-grams, through which the vector representation for an OOV word can be obtained by simply summing over the representations of its n-grams.

**POS Features.** (Nallapati et al., 2016) included POS in their feature-rich encoder. They created additional look-up based embedding matrices for the set of POS tags, named-entity tags, similar to the embeddings for words, to identify the key concepts and key entities in the sentence. (Simon & Kešelj, 2018) also presented a method for automatic term extraction for the technical domain based on the use of part-of-speech features and common words.

## 7. Conclusions

Through the experiments, we can conclude that combining POS and character-level features could improve the performance in the abstractive text summarization task, however, the improvement might not be so significant as reported by other works focusing on using high-level features such as sentences (Hsu et al., 2018), (Tan et al., 2017). We also find that different methods of integrating features can have different effect. While all the proposed models in our work can improve the performance in some extent, the model with single-encoder + character features attains the largest improvement.

Based on our analysis, the models with character-level features have a higher improvement on the documents with more rare words and even OOV words, which may indicate that our model has a better understanding in rare words with the help of character-level features. Although our models with POS or Char features almost produce the same summaries as the baseline, in some cases, they capture the exact word in the reference, and thus score slightly higher in terms of ROUGE. Moreover, the ability of our models being abstractive is also limited, because they can only generate new words in a limited set of common words from sports news.

Due to the limitation of time and computational resources, we have not done further explorations on the systematic differences between the single-encoder and multi-encoder architectures. Future work will focus on the systematic comparison between these encoder architectures, as well as the further discussion about how the specific mechanism of POS and character-level features interact with pointer generator architecture.

## References

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bird, Steven. Nltk: The natural language toolkit. In *COLINGâĂć ACL 2006*, pp. 69, 2006.

Bojanowski, Piotr, Grave, Edouard, Joulin, Armand, and Mikolov, Tomas. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Chung, Tong Lee, Xu, Bin, Liu, Yongbin, and Ouyang, Chunping. Main point generator: Summarizing with a focus. In Pei, Jian, Manolopoulos, Yannis, Sadiq, Shazia, and Li, Jianxin (eds.), *Database Systems for Advanced Applications*, pp. 924–932, Cham, 2018. Springer International Publishing. ISBN 978-3-319-91452-7.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Fan, Angela, Grangier, David, and Auli, Michael. Controllable abstractive summarization. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pp. 45–54, 2018.

Gehring, Jonas, Auli, Michael, Grangier, David, Yarats, Denis, and Dauphin, Yann N. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1243–1252. JMLR. org, 2017.

Gulcehre, Caglar, Ahn, Sungjin, Nallapati, Ramesh, Zhou, Bowen, and Bengio, Yoshua. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of*

*the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 140–149, 2016.

Hermann, Karl Moritz, Kocisky, Tomas, Grefenstette, Edward, Espeholt, Lasse, Kay, Will, Suleyman, Mustafa, and Blunsom, Phil. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pp. 1693–1701, 2015.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hsu, Wan-Ting, Lin, Chieh-Kai, Lee, Ming-Ying, Min, Kerui, Tang, Jing, and Sun, Min. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 132–141, 2018.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Liang, Dongyun, Xu, Weiran, and Zhao, Yinge. Combining word-level and character-level representations for relation classification of informal text. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pp. 43–47, 2017.

Lin, Chin-Yew. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

Marcus, Mitchell P, Santorini, Beatrice, and Marcinkiewicz, Mary Ann. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2), 1993.

Mikolov, Tomáš, Karafiát, Martin, Burget, Lukáš, Černockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Nallapati, Ramesh, Zhou, Bowen, dos Santos, Cicero, Gulcehre, Caglar, and Xiang, Bing. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280–290, 2016.

Paulus, Romain, Xiong, Caiming, and Socher, Richard. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

Ranzato, Marc'Aurelio, Chopra, Sumit, Auli, Michael, and Zaremba, Wojciech. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Rush, Alexander M, Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 379–389, 2015.

See, Abigail, Liu, Peter J, and Manning, Christopher D. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 1073–1083, 2017.

Simon, Nisha Ingrid and Kešelj, Vlado. Automatic term extraction in technical domain using part-of-speech and common-word features. In *Proceedings of the ACM Symposium on Document Engineering 2018*, pp. 51. ACM, 2018.

Tan, Jiwei, Wan, Xiaojun, and Xiao, Jianguo. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 1171–1181, 2017.

Tu, Zhaopeng, Lu, Zhengdong, Liu, Yang, Liu, Xiaohua, and Li, Hang. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 76–85, 2016.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V, Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. 2016.

Zhang, Xingxing and Lapata, Mirella. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 584–594, 2017.