Stick to Facts: Towards Fidelity-oriented Product Description Generation

Zhangming Chan\textsuperscript{1,2,∗,†}, Xiuying Chen\textsuperscript{1,2,∗}, Yongliang Wang\textsuperscript{3}, Juntao Li\textsuperscript{1,2}, Zhiqiang Zhang\textsuperscript{3}, Kun Gai\textsuperscript{3}, Dongyan Zhao\textsuperscript{1,2} and Rui Yan\textsuperscript{1,2,‡}

\textsuperscript{1} Center for Data Science, AAIS, Peking University
\textsuperscript{2} Wangxuan Institute of Computer Technology, Peking University
\textsuperscript{3} Alibaba Group, Beijing

\{zhangming.chan, xy-chen, ljt, zhaody, ruiyan\}@pku.edu.cn
\{yongliang.wyl, zhang.zhiqiang, jingshi.gk\}@alibaba-inc.com

Abstract

Different from other text generation tasks, in product description generation, it is of vital importance to generate faithful descriptions that stick to the product attribute information. However, little attention has been paid to this problem. To bridge this gap we propose a model named Fidelity-oriented Product Description Generator (FPDG). FPDG takes the entity label of each word into account, since the product attribute information is always conveyed by entity words. Specifically, we first propose a Recurrent Neural Network (RNN) decoder based on the Entity-label-guided Long Short-Term Memory (ELSTM) cell, taking both the embedding and the entity label of each word as input. Second, we establish a keyword memory that stores the entity labels as keys and keywords as values, and FPDG will attend to keywords through attending to their entity labels. Experiments conducted a large-scale real-world product description dataset show that our model achieves the state-of-the-art performance in terms of both traditional generation metrics as well as human evaluations. Specifically, FPDG increases the fidelity of the generated descriptions by 25%.

1 Introduction

The effectiveness of automatic text generation has been proved in various natural language processing applications, such as neural machine translation (Luong et al., 2015; Zhou et al., 2017), dialogue generation (Tao et al., 2018a; Hu et al., 2019), and abstractive text summarization (Chen et al.; Gao et al., 2019). One such task, product description generation has also attracted considerable attention (Lipton et al., 2015; Wang et al., 2017; Zhang et al., 2019a). An accurate and attractive description not only helps customers make an informed decision but also improves the likelihood of purchase. Concretely, the product description generation task takes several keywords describing the attributes of a product as input, and then outputs fluent and attractive sentences that highlight the feature of this product.

There is one intrinsic difference between product description generation and other generation tasks, such as story or poem generation (Li et al., 2018; Yao et al., 2019); the generated description has to be faithful to the product attributes. An example case is shown in Table 1, where the good product description matches the input information, while the bad description mistakes the brand and the style of the jeans. Our preliminary study reveals that 48% of the outputs from a state-of-the-art sequence-to-sequence system suffer from this problem. In the real-world e-commerce product description generation system, generating text with unfaithful information is unacceptable. A wrong brand name will damage the interests of advertisers, while a wrong product category might break the law by misleading consumers. Any kind of unfaithful description will bring huge economic losses to online platforms.

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|}
\hline
Input & high-waist; straight; jeans; blue; ZARA \\
\hline
Bad output & This UNIQLO low-waist jeans are suitable for the curvature of the waistline and will not slip when worn for a long time. The straight version and black color together makes you even slimmer. \\
\hline
Good output & This blue jeans look personalized and chic, making you more distinctive. With a high-waist design, the legs look even more slender, and the whole person looks taller. The straight design hides the fat in the legs and is visually slimmer. \\
\hline
\end{tabular}
\end{center}
\caption{Example of product description generation. The text in red demonstrates faithful description, and text in blue demonstrates wrong description.}
\end{table}
Though of great importance, no attention has been paid to this problem. Existing product description generators include (Chen et al., 2019; Zhang et al., 2019b), where they focus on generating personalized descriptions and pattern-controlled descriptions, respectively.

In this paper, we address the fidelity problem in generation tasks by developing a model named Fidelity-oriented Product Description Generator (FPDG), which takes keywords about the product attributes as inputs. Since product attribute information is always conveyed by entity words (up to 89.72% in input keywords), FPDG generates faithful product descriptions by taking into account the entity label of each word. Specifically, first, we propose an Entity-label-guided Long Short-Term Memory (ELSTM) as the cell in the decoder RNN, which takes the entity label of an input word, as well as the word itself as input. Then, we establish a keyword memory storing the input keywords with their entity labels. In each decoding step, the current hidden state of the ELSTM focuses on proper words in this keyword memory in regard to their entity categories. We also collect a large-scale real-world product description dataset from one of the largest e-commerce platforms in China. Extensive experiments conducted on this dataset show that FPDG outperforms the state-of-the-art baselines in terms of traditional generation metrics and human evaluations. Specifically, FPDG greatly improves the fidelity of generated description by 24.61%.

To our best knowledge, we are the first to explore the fidelity problem of product description generation. Besides, to tackle this problem, we propose an ELSTM and a keyword memory to incorporate entity label information, so as to generate more accurate descriptions.

2 Related Work

We detail related work on text generation, entity-related generation, and product description.

Text generation. Recently, sequence-to-sequence (Seq2Seq) neural network models have been widely used in NLG approaches. Their effectiveness has been demonstrated in a variety of text generation tasks, such as neural machine translation (Luong et al., 2015; Bahdanau et al., 2014; Wu et al., 2016), abstractive text summarization (See et al., 2017a; Hsu et al., 2018; Chen and Bansal, 2018), dialogue generation (Tao et al., 2018a; Xing et al., 2017), etc. Along another line, there are also works based on an attention mechanism. Vaswani et al. (2017) proposed a Transformer architecture that utilizes the self-attention mechanism and has achieved state-of-the-art results in neural machine translation. Since then, the attention mechanism has been used in a variety of tasks (Devlin et al., 2018; Fan et al., 2018; Zhou et al., 2018).

Entity-related generation. Named entity recognition (NER) is a fundamental component in language understanding and reasoning (Greenberg et al., 2018; Katiyar and Cardie, 2018). In (Ji et al., 2017), they proved that adding entity-related information can improve the performance of language modeling. Building upon this work, in (Clark et al., 2018), they combined entity context with previous-sentence context, and demonstrated the importance of the latter in coherence test. Another line of related work generates recipes using neural networks to track and update entity representations (Bosselut et al., 2018). Different from the above works, we utilize entity labels as supplementary information to assist decoding in the text generation task.

Product descriptions. Quality product descriptions are critical for providing a competitive customer experience in an e-commerce platform. Due to its importance, automatically generating the product description has attracted considerable interests. Initial works include (Wang et al., 2017), which incorporates statistical methods with the template to generate product descriptions. With the development of neural networks, (Chen et al., 2019) explored a new way to generate personalized product descriptions by combining the power of neural networks and a knowledge base. (Zhang et al., 2019b) proposed a pointer-generator neural network to generate product description whose patterns are controlled.

In real-world product description generation application, however, the most important prerequisite is the fidelity of generated text, and to the best of our knowledge, no research has been conducted on this so far.

3 Problem Formulation

FPDG takes a list of keywords $X = \{x_1, ..., x_{T_X}\}$ as inputs, where $T_X$ is the number of keywords. These keywords are all about the important attributes of the product. The goal of FPDG is to
generate a product description \( \hat{Y} = \{\hat{y}_1, \ldots, \hat{y}_{T_y}\} \) that is not only grammatically correct but also consistent with the input information, such as the brand name and the fashion style. Essentially, FPDG tries to optimize the parameters to maximize the probability \( P(Y|X) = \prod_{t=1}^{T_y} P(y_t|X) \), where \( Y = \{y_1, \ldots, y_{T_y}\} \) is the ground truth answer.

4 Model

In this section, we introduce our Fidelity-oriented Product Description Generator (FPDG) model in detail. An overview of FPDG is shown in Figure 1 and can be split into two modules:

(1) **Keyword Encoder** (See § 4.1): We first use a key-value memory to store the entity label as key and the corresponding word as value. To better learn the interaction between words, we employ two self-attention modules from Transformer (Vaswani et al., 2017) to model the input keywords and their entity-labels separately.

(2) **Entity-based Generator** (See § 4.2): We propose a recurrent decoder based on Entity-label-guided LSTM (ELSTM) to generate the product description.

4.1 Keyword Encoder

In the product description dataset, most of the input keywords are entity words (up to 89.72%), and the description text should be faithful to these entity words. Hence, we incorporate entity label information to improve the accuracy of the generated text. In this section, we introduce how to embed input keywords with entity label information.

To begin with, we use an embedding matrix \( e \) to map a one-hot representation of each word in \( x_i \) into a high-dimensional vector space. Since our input keywords have no order information, we use the Self-Attention Module (SAM) from Transformer (Vaswani et al., 2017) to model the temporal interactions between the words, instead of RNN-based encoder. We use three fully-connected layers to project \( e(x_i) \) into three spaces, i.e., the query \( q_i = F_q(e(x_i)) \), the key \( k_i = F_k(e(x_i)) \) and the value \( v_i = F_v(e(x_i)) \). The attention module then takes \( q_i \) to attend to each \( k_i \), and uses these attention distribution results \( \alpha_{i,j} \in \mathbb{R}^{Tx} \) as weights to obtain the weighted sum of \( v_i \), as shown in Equation 2. Next, we add the original word representation \( e(x_i) \) on \( \beta_i \) as the residential connection layer, as shown in Equation 3:

\[
\alpha_{i,j} = \frac{\exp(q_i k_j)}{\sum_{n=1}^{T_x} \exp(q_i k_n)}, \quad (1)
\]

\[
\beta_i = \sum_{j=1}^{T_x} \alpha_{i,j} v_j, \quad (2)
\]

\[
h_i = e(x_i) + \beta_i, \quad (3)
\]

where \( \alpha_{i,j} \) denotes the attention weight of the \( i \)-th word on the \( j \)-th word. Finally, we apply a feed-forward layer on \( h_i \) to obtain the final word representation \( \hat{h}_i \):

\[
h_i = \text{max}(0, \hat{h}_i \cdot W_1 + b_1) \cdot W_2 + b_2, \quad (4)
\]

where \( W_1, W_2, b_1, b_2 \) are all trainable parameters. We refer to the above process as:

\[
h_i = \text{SAM}(F_q(x_i), F_k(x_i), F_v(x_i)). \quad (5)
\]

In the meantime, we leverage the e-commerce-adapted AliNER\(^1\) to label each word in the input such as “brand name” and “color”. Non-entity words are labeled as “normal word”. We denote \( c_i \) as the entity label for the \( i \)-th word. To better learn

---

\(^1\)https://ai.aliyun.com/nlp/ner
the interaction between these entity labels, we use a second SAM, taking $e$ as input, and obtain the final representation for $c_i$ as $m_i$:

$$m_i = \text{SAM}(F_q(c_i), F_k(c_i), F_v(c_i)).$$

We also propose a key-value keyword memory that stores the label results as shown in Figure 1. The keys in the keyword memory are $T_C$ representations of different entity-labels, where $T_C$ is the number of entity categories. The values in the memory are self-attention representations of words that belong to each category. We denote the $k$-th entity-label in the keyword memory as $e(x^k_i)$, and the $i$-th word belonging to this category as $e(x^k_i)$. This keyword memory will be incorporated into the generation process in §4.2.3.

4.2 Entity-based Generator

To incorporate entity label information into generation process, we propose a modified version of LSTM, named Entity-label-guided Long Short-Term Memory (ELSTM). We first introduce ELSTM and then introduce the RNN decoder based on ELSTM.

4.2.1 ELSTM

As shown in Figure 3, ELSTM consists of three LSTMs, two Word-LSTMs and one Entity-label-LSTM. The hidden states of the two Word-LSTMs are integrated together, forming $w^h_t$, while the hidden state of Entity-Label-LSTM is $l^h_t$. The word hidden state $w^h_t$ attends to the SAM outputs to predict the next word, while the entity label hidden state $l^h_t$ is used to attend to the keyword memory and predict entity label of next word. In other words, ELSTM is a hybrid of three LSTMs with a deeper interaction between these cells. Overall, ELSTM takes two variables as input, the embedding of an input word $e(y_t)$ and its entity label $m_t$.

The structure of each LSTM in ELSTM is the same as original LSTM, thus, is omitted here due to space limitations. Next, we introduce the interaction in ELSTM in detail. As shown in Figure 2, Word-LSTM$_0$ takes the $e(y_t)$ as input, and outputs the initial word hidden state $w^h_{t+1}$:

$$w^h_{t+1} = \text{Word-LSTM}_0(w^h_t, e(y_t)).$$

Next we calculate the entity label hidden state by Entity-Label-LSTM, taking both $m_t$ and $w^h_{t+1}$ as input:

$$l^h_{t+1} = \text{Entity-Label-LSTM}(l^h_t, m_t + w^h_{t+1}).$$

To further improve the accuracy of entity label prediction, we also apply another multi-layer projection, incorporating entity label context vector $c^m_t$ (introduced in §16) to obtain polished entity hidden state $l^h_{t+1}$:

$$l^h_{t+1} = FC([l^h_t, e^m_t]),$$

where $[,]$ denotes the concatenation operation.

Finally, Word-LSTM$_1$ takes the entity-label hidden state $l^h_{t+1}$ and word embedding $e(y_t)$ as input, and outputs predicted word hidden state that contains entity label information:

$$w^h_{t+1} = \text{Word-LSTM}_1(w^h_t, e(y_t) + l^h_{t+1}).$$

Note that it is possible that the predicted entity-label hidden state is not accurate enough. Hence, we apply a gate fusion combining $w^h_{t+1}$ with initial word hidden state $w^h_{t+1'}$ to ensure the word hidden state quality:

$$\gamma = \sigma(FC([w^h_{t+1}, w^h_{t+1'}])),$$

$$w^h_{t+1} = \gamma w^h_{t+1'} + (1 - \gamma) w^h_{t+1'}.$$

The fusion result is $w^h_{t+1}$, i.e., the updated word hidden state.

In this way, the entity label information and word information are fully interacted in ELSTM cell, while the hidden states are updated.

4.2.2 ELSTM-based Attention Mechanism

Established on ELSTMs, we now have a new RNN decoder to generate descriptions, incorporating the two SAM encoders and the keyword memory. First, we apply an Bi-RNN to encode input
keywords, and use its last hidden state as decoder initial hidden states, i.e., \( w_0^h \) and \( l_0^h \). The \( t \)-th decoding step is calculated as:

\[
\begin{align*}
[w_{t+1}^h | l_{t+1}^h] & = \text{ELSTM} \left( [w_t^h | l_t^h], (e(y_t), m_t) \right), \quad (13)
\end{align*}
\]

Next, similar to the traditional attention mechanism from (Bahdanau et al., 2015), we summarize the input word representation \( e(y_t) \) and entity label representation \( m \) into the word context vector \( c_t^w \) and entity label vector \( c_t^m \), respectively. As for how to obtain \( c_t^w \) and \( c_t^m \), we use the two hidden states in ELSTM, i.e., \( w_t^h \) and \( l_t^h \) to attend to \( h_t \) and \( m_t \), respectively. Specifically, the entity label context vector \( c_t^m \) is calculated as:

\[
\begin{align*}
\gamma_{t,i} & = W_\gamma \tanh \left( W_\nu h_i + W_h h_i \right), \quad (14) \\
\gamma_{t,i} & = \exp \left( \gamma_{t,i} / \sum_{j=1}^{T_X} \exp \left( \gamma_{t,j} \right) \right), \quad (15) \\
c_{t,i}^m & = \sum_{i=1}^{T_x} \gamma_{t,i} m_i. \quad (16)
\end{align*}
\]

The decoder state \( l_{t+1}^h \) is used to attend to each entity label representation \( m_i \), resulting in the attention distribution \( \gamma_t \in \mathbb{R}^{T_x} \), as shown in Equation 15. Then we use the attention distribution \( \gamma_t \) to obtain a weighted sum of the entity label representations \( m \) as the word context vector \( c_t^w \), shown in Equation 16. \( c_t^m \) is obtained in a similarity by using hidden state \( w_t^h \) attending to \( e(y_t) \), and we omit the details for brevity. This two context vectors play different parts, and is introduced in §4.2.4.

### 4.2.4 Projection Layers

Next, using a fusion gate \( g_t \), \( o_{t+1} \) is combined with word hidden state \( c_{t+1}^w \) in a similar way in Equation 12 to obtain \( o_{t+1} \). Finally, we obtain the final generation distribution \( P_v \) over vocabulary:

\[
P_{t+1}^v = \text{softmax} \left( W_v [o_{t+1}, w_{t+1}^h] + b_v \right). \quad (20)
\]

We concatenate the memory vector \( o_{t+1} \), the word context vector \( c_{t+1}^m \), and the output of the decoder \( w_{t+1}^h \) as the input of the output projection layer.

Apart from predicting the next word, we also use the entity label hidden state \( l_t^h \) to predict the entity label of the next word as an auxiliary task. The distribution \( P_e^{t+1} \) over entity categories is calculated as:

\[
P_{t+1}^e = \text{softmax} (W_e [l_{t+1}^h, c_{t+1}^m] + b_e) \quad (21)
\]

We use negative log-likelihood as loss function:

\[
\mathcal{L} = - (\sum_{t=1}^{T_y} \log P_v(y_t) + \lambda \cdot \sum_{t=1}^{T_y} \log P_e(m_t)), \quad (22)
\]

where \( \lambda \) is the weight of entity label prediction loss. The gradient descent method is employed to update all parameters and minimize this loss function.
5 Experimental Setup

5.1 Research Questions

We list four research questions that guide the experiments: **RQ1** (See § 6.1): What is the overall performance of FPDG? Are generated descriptions faithful? **RQ2** (See § 6.2): What is the effect of each module in FPDG? **RQ3** (See § 6.3): How does the keyword memory work in each decoding step? **RQ4** (See § 6.4): Can ELSTM successfully capture entity label information?

5.2 Dataset

We collect a large-scale real-world product description dataset from one of the largest e-commerce platforms in China\(^2\). The inputs are keywords about the product attributes selected by sellers on the platform, and the product descriptions are written by professional experts that are good at marketing. Overall, there are 404,000 training samples, and 5,000 validation and test samples. On average, there are 10 keywords in the input, and 63 words in the product description. 89.72% inputs are entity words.

5.3 Evaluation Metrics

Following (Fu et al., 2017), we use the evaluation package of (Chen et al., 2015), which includes BLEU-1, BLEU-2, BLEU-3, BLEU-4, METEOR and ROUGE-L. BLEU is a popular machine translation metric that analyzes the co-occurrences of n-grams between the candidate and reference sentences. METEOR is calculated by generating an alignment between the words in the candidate and reference sentences, with an aim of 1:1 correspondence. ROUGE-L is a metric designed to evaluate text summarization algorithms.

(Tao et al., 2018b) notes that only using the BLEU metric to evaluate text quality can be misleading. Therefore, we also evaluate our model by human evaluation. Three highly educated participants are asked to score 100 randomly sampled summaries generated by Pointer-Gen and FPDG. We chose Pointer-Gen since its performance is relatively higher than other baselines. The statistical significance of observed differences between the performance of two runs are tested using a two-tailed paired t-test and is denoted using \(\dagger\) (or \(\ddagger\)) for strong significance for \(\alpha = 0.01\).

5.4 Comparison Methods

We first conduct an ablation study to prove the effectiveness of each module in FPDG. Model w/o ELSTM is implemented with only keys in the memory because there are no entity representations as values. Then, to evaluate the performance of our proposed dataset and model, we compare it with the following baselines:

(1) **Seq2Seq**: The sequence-to-sequence framework (Sutskever et al., 2014) is one of the initial works proposed for the language generation task.

(2) **Pointer-Gen**: A sequence-to-sequence framework with pointer and coverage mechanism proposed in (See et al., 2017b).

(3) **Conv-Seq2seq**: A model combining the convolutional neural networks and the sequence-to-sequence network (Gehring et al., 2017).

(4) **FTSum**: A faithful summarization model proposed in (Cao et al., 2018), which leverages open information extraction and dependency parse technologies to extract actual fact descriptions from the source text.

(5) **Transformer**: A network architecture that solely based on attention mechanisms, dispensing with recurrence and convolutions entirely (Vaswani et al., 2017).

(6) **PCPG**: A pattern-controlled product description generation model proposed in (Zhang et al., 2019b). We adapt the model for our scenario.

To verify whether the performance improvement is obtained by adding additional entity label inputs, we directly concatenate the word embedding with the entity embedding as input for these baselines, denoted as “with Entity Embedding” in Table 2.

5.5 Implementation Details

We implement our experiments in Pytorch\(^3\) on Tesla V100 GPUs\(^4\). The word and entity-label embedding dimensions are set to 256. The number of hidden units and the entity-label hidden size are also set to 256. All inputs were padded with zeros to a maximum keyword number of the batch. There are 36 categories of entity labels together. \(\lambda\) is set to 0 in the first 500 steps, and 0.6 in the rest of the training process. We performed mini-batch cross-entropy training with a batch size of 256 documents for 15 training epochs. We set the

\(^2\)https://www.taobao.com/

\(^3\)https://pytorch.org/

\(^4\)Our model will be incorporated into Chuangyi-Taobao (https://chuangyi.taobao.com/pages/smartPhrase).
Table 2: RQ1: Comparison between baselines.

<table>
<thead>
<tr>
<th>Models</th>
<th>BLEU Metric</th>
<th>METEOR</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLEU-1</td>
<td>BLEU-2</td>
<td>BLEU-3</td>
</tr>
<tr>
<td>Seq2seq</td>
<td>30.10</td>
<td>13.61</td>
<td>7.564</td>
</tr>
<tr>
<td>with Entity Embedding</td>
<td>30.46</td>
<td>13.77</td>
<td>7.722</td>
</tr>
<tr>
<td>Pointer-Gen</td>
<td>30.66</td>
<td>13.84</td>
<td>7.933</td>
</tr>
<tr>
<td>Conv-Seq2Seq</td>
<td>28.22</td>
<td>12.15</td>
<td>6.393</td>
</tr>
<tr>
<td>with Entity Embedding</td>
<td>28.23</td>
<td>12.22</td>
<td>6.401</td>
</tr>
<tr>
<td>FTSum</td>
<td>30.33</td>
<td>13.57</td>
<td>7.563</td>
</tr>
<tr>
<td>with Entity Embedding</td>
<td>30.47</td>
<td>13.61</td>
<td>7.611</td>
</tr>
<tr>
<td>Transformer</td>
<td>29.11</td>
<td>12.77</td>
<td>7.116</td>
</tr>
<tr>
<td>with Entity Embedding</td>
<td>29.37</td>
<td>12.64</td>
<td>6.852</td>
</tr>
<tr>
<td>PCPG(including Entity)</td>
<td>29.46</td>
<td>13.02</td>
<td>7.259</td>
</tr>
<tr>
<td>Our FPDG</td>
<td>32.26</td>
<td>14.79</td>
<td>8.472</td>
</tr>
<tr>
<td>w/o KW-MEM</td>
<td>31.85</td>
<td>14.36</td>
<td>8.134</td>
</tr>
<tr>
<td>w/o ELSTM</td>
<td>31.41</td>
<td>14.57</td>
<td>8.430</td>
</tr>
</tbody>
</table>

Table 3: RQ1: Human evaluation comparison with Pointer-Gen baseline.

<table>
<thead>
<tr>
<th></th>
<th>Fluency</th>
<th>Informativity</th>
<th>Fidelity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer-Gen</td>
<td>2.23</td>
<td>1.84</td>
<td>1.91</td>
</tr>
<tr>
<td>FPDG</td>
<td>2.46*</td>
<td>2.19*</td>
<td>2.38*</td>
</tr>
</tbody>
</table>

minimum encoding step to 15 and maximum decoding step size to 70. During decoding, we employ beam search with a beam size of 4 to generate a more fluent sentence. It took around 6 hrs on GPUs for training. After each epoch, we evaluated our model on the validation set and chose the best performing model for the test set. We use the Adam optimizer (Duchi et al., 2010) as our optimizing algorithm and the learning rate is 1e-3.

6 Experimental Results

6.1 Overall Performance

For research question RQ1, we examine the performance of our model and baselines in terms of BLEU, as shown in Table 2. Firstly, among all baselines, Pointer-Gen obtains the best performance, outperforming the worst baseline Conv-Seq2Seq by 2.44 in BLEU-1. Secondly, directly concatenating the entity label embedding with the word embedding does not bring much help, only leading to an improvement of 0.26 in BLEU-1 for the Pointer-Gen model. Finally, our model outperforms all baselines for all metrics, outperforming the strongest baseline Pointer-Gen, by 5.22%, 6.86%, 6.79%, and 6.65% in terms of BLEU-1, BLEU-2, BLEU-3, and BLEU-4, respectively.

As for human evaluation, we ask three highly educated participants to rank generated summaries in terms of fluency, informativity, and fidelity. The rating score ranges from 1 to 3 with 3 being the best. The results are shown in Table 3, where FPDG outperforms Pointer-Gen by 10.31% and 19.02% in terms of fluency and informativity, and, specifically, FPDG greatly improves the fidelity value by 24.61%. We also conduct the paired student t-test between our model and Pointer-Gen, and the result demonstrates the significance of the above results. The kappa statistics are 0.55 and 0.49, respectively, which indicates fair and moderate agreement between annotators\(^5\).

\(^5\)(Landis and Koch, 1977) characterize kappa values < 0 as no agreement, 0-0.20 as slight, 0.21-0.40 as fair, 0.41-0.60 as moderate, 0.61-0.80 as substantial, and 0.81-1 as almost perfect agreement.
6.2 Ablation Study

Next, we turn to research question RQ2. We conduct ablation tests on the usage of the keyword memory and ELSTM, corresponding to FPDG w/o KW-MEM and ELSTM, respectively. The ROUGE score result is shown at the bottom of Table 2. Performances of all ablation models are worse than that of FPDG in terms of almost all metrics, which demonstrates the necessity of each module in FPDG. Specifically, ELSTM makes the greatest contribution to FPDG, improving the BLEU-1, BLEU-2 scores by 2.71% and 1.51%.

6.3 Analysis of Keyword Memory

We then address RQ3 by analyzing the entity-label-level attention on the keyword memory. Two representative cases are shown in Figure 4. The figure in the above is the attention map when generating the word “toryburch”, and the bottom figure is when generating the word “printflower”. The darker the color, the higher the attention. Due to limited space, we omit irrelevant entity categories. When generating “toryburch”, which is a brand name, the entity-label-level attention pays most attention to the “Brand” entity label, and when generating “flower”, which is a style element, it most pays attention to “Element”. This example demonstrates the effectiveness of the entity-label-level attention.

6.4 Analysis of ELSTM

We now turn to RQ4: whether or not the ELSTM can capture entity label information. We examine this question by verifying whether ELSTM can predict the entity label of the generated word. The accuracy of the predicted entity label is calculated in a teacher-forcing style, i.e., each ELSTM takes the ground truth entity label and word as input, and outputs the entity label of the next word. We employ recall at position $k$ in $n$ candidates ($R_k@n$) as evaluation metrics. Over the whole test dataset, $R_1@36$ is 64.12%, $R_2@36$ is 80.86%, and $R_3@36$ is 94.02%, which means ELSTM can capture the entity label information to a great extent and guide the word generation.

We also show a case study in Table 4. The description generated by Pointer-Gen introduces the Prada bag as a “clamshell bag has a good anti-theft effect”, which is contrary to the fact. While our model generates the faithful description: “The opening and closing design of the zipper smoothes the stroke”.

7 Conclusion and Future Work

In this paper, we explore the fidelity problem in product description generation. To tackle this challenge, based on the consideration that product attribute information is typically conveyed by...
entity words, we incorporate the entity label information of each word to enable the model to have a better understanding of the words and better focus on key information. Specifically, we propose an Entity-label-guided Long Short-Term Memory (ELSTM) and a token memory to store and capture the entity label information of each word. Our model outperforms state-of-the-art methods in terms of BLEU and human evaluations by a large margin. In the near future, we aim to fully prevent the generation of unfaithful descriptions and bring FPDG online.

Acknowledgments

We would like to thank the reviewers for their constructive comments. This work was supported by the National Key Research and Development Program of China (No. 2017YFC0804001), the National Science Foundation of China (NSFC No. 61876196 and NSFC No. 61672058). Rui Yan was sponsored by Alibaba Innovative Research (AIR) Grant. Rui Yan is the corresponding author.

References


Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In ICLR.

Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. ICLR.

Ziqiang Cao, Furu Wei, Wenjie Li, and Sujian Li. 2018. Faithful to the original: Fact aware neural abstractive summarization. In AAAI.


Xiuying Chen, Zhangming Chan, Shen Gao, Meng-Hsuan Yu, Dongyan Zhao, and Rui Yan. Learning towards abstractive timeline summarization. In IJCAI.


