Generating Diverse Numbers of Diverse Keyphrases

Xingdi Yuan *
Microsoft Research Montréal
Montréal, Québec, Canada
eric.yuan@microsoft.com

Tong Wang *
Microsoft Research Montréal
Montréal, Québec, Canada
tong.wang@microsoft.com

Rui Meng *
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, 15213
rui.meng@pitt.edu

Khushboo Thaker
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, 15213
k.thaker@pitt.edu

Daqing He
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, 15213
dah44@pitt.edu

Adam Trischler
Microsoft Research Montréal
Montréal, Québec, Canada
adam.trischler@microsoft.com

Abstract

Existing keyphrase generation studies suffer from the problems of generating duplicate phrases and deficient evaluation based on a fixed number of predicted phrases. We propose a recurrent generative model that generates multiple keyphrases sequentially from a text, with specific modules that promote generation diversity. We further propose two new metrics that consider a variable number of phrases. With both existing and proposed evaluation setups, our model demonstrates superior performance to baselines on three types of keyphrase generation datasets, including two newly introduced in this work: StackExchange and TextWorld ACG. In contrast to previous keyphrase generation approaches, our model generates sets of diverse keyphrases of a variable number.

1 Introduction

Keyphrases are short pieces of text that humans use to summarize the high-level meaning of a longer text, or to highlight certain important topics or information. Keyphrase generation is the task of automatically predicting keyphrases given a source text. Models that perform this task should be capable not only of distilling high-level information from a document, but also of locating specific, important snippets within it. Complicating the problem, keyphrases may or may not appear directly and verbatim in their source text (they may be *present* or *absent*).

A given source text is usually associated with a *set* of keyphrases. Thus, keyphrase generation is an instance of set generation, where each element in the set is a short sequence of tokens and the size of the set varies depending on the source. Most prior studies approach keyphrase generation similarly to summarization, relying on sequence-to-sequence (Seq2Seq) methods (Meng et al. (2017); Chen et al. (2018a); Ye and Wang (2018); Chen et al. (2018b)). Conditioned on a source text, Seq2Seq models generate phrases individually or as a longer, concatenated sequence with delimiting tokens throughout. Standard Seq2Seq models generate only one sequence at a time. To overcome this

*These authors contributed equally. The order is determined by a fidget spinner.

limitation so that a sufficient set of diverse phrases can be generated, one common approach is to use beam-search decoding with a fixed, large beam width. Models are then evaluated by taking the top k results from the over-generated beam of phrases (k is typically 5 or 10) and comparing them to "groundtruth" keyphrases.

Though this approach has achieved good results, we argue that it suffers from two major problems. Firstly, the evaluation setup is suboptimal because of the mismatch between a fixed k and the number of groundtruth keyphrases for a text. The appropriate number of keyphrases for each text can vary drastically, depending on factors like the length or topic of the text or the granularity of keyphrase annotation. Therefore, arbitrarily using the same k to evaluate on all data samples may not be appropriate. With the existing evaluation setup, for example, we find that the upper bounds for $F_1@5$ and $F_1@10$ on KP20K is 0.858 and 0.626, respectively (see Section 4.1 for more details), and worse for datasets where fewer keyphrases are available. Secondly, the beam-search strategy ignores interactions between generated phrases. This often results in insufficient diversity in decoded phrases. Although models such as in Chen et al. (2018a) or Ye and Wang (2018) can take diversity into account during training, they rarely achieve it during decoding since they must over-generate and rank phrases with beam search.

To overcome the above issues, we propose two improvements with regard to decoding and evaluation for keyphrase generation frameworks. First, we propose a novel keyphrase generation model to fit the demand of generating variable numbers of diverse phrases. This model predicts the optimal number of phrases to generate for a given text, and uses a target encoder and orthogonal regularization to facilitate more diverse phrase generation. Second, we propose two variable numbers, $\mathcal{M}$ and $\mathcal{V}$, in the evaluation as the cutoff for computing scores such as F_1. They show better empirical characteristics than previous metrics based on a fixed k. Besides the two improvements in modeling and evaluation, a third major contribution of our study is two brand-new datasets for keyphrase generation: STACKEXCHANGE and TEXTWORLD ACG. Because their source material is distinct from scientific publications as used in previous corpora, we expect these datasets to contribute to a more comprehensive testbed for keyphrase generation.

2 Related Work

2.1 Keyphrase Extraction and Generation

Traditional keyphrase extraction has been studied extensively in past decades. In most existing literature, keyphrase extraction has been formulated as a two-step process. First, lexical features such as part-of-speech tags are used to determine a list of phrase candidates by heuristic methods (Witten et al. (1999); Liu et al. (2011); Wang et al. (2016); Yang et al. (2017)). Second, a ranking algorithm is adopted to rank the candidate list and the top ranked candidates are selected as keyphrases. A wide variety of methods were applied for ranking, such as bagged decision trees (Medelyan et al., 2009; Lopez and Romary, 2010), Multi-Layer Perceptron and Support Vector Machine (Lopez and Romary, 2010) and PageRank ((Mihalcea and Tarau, 2004; Le et al., 2016; Wan and Xiao, 2008)). Recently, Zhang et al. (2016); Luan et al. (2017); Gollapalli et al. (2017) used sequence labeling models to extract keyphrases from text. Similarly, Subramanian et al. (2017) used Pointer Networks to point to the start and end positions of keyphrases in a source text.

The main drawback of keyphrase extraction is that sometimes keyphrases are absent from the source text, thus an extractive model will fail predicting those keyphrases. Meng et al. (2017) first proposed the CopyRNN, a neural generative model that both generates words from vocabulary and points to words from the source text. Recently, based on the CopyRNN architecture, Chen et al. (2018a) proposed CorrRNN, which takes states and attention vectors from previous steps into account in both encoder and decoder to reduce duplication and improve coverage. Ye and Wang (2018) proposed semi-supervised methods by leveraging both labeled and unlabeled data for training. Chen et al. (2018b); Ye and Wang (2018) proposed to use structure information (e.g., title of source text) to improve keyphrase generation performance. However, none of the above models have the ability to generate variable numbers of keyphrases.

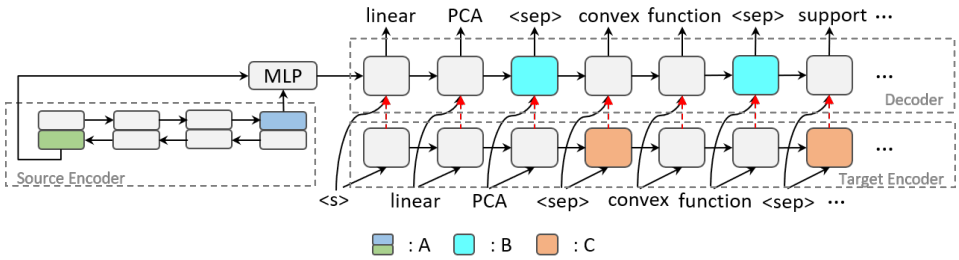

Figure 1: Overall structure of our proposed model. A represents last states of the bi-directional source encoder; B represents decoder states where target tokens are delimiters; C indicates target encoder states where input tokens are delimiters. During orthogonal regularization, all B states are used; during target encoder training, we maximize the mutual information between states A with each of the C states; red dash arrow indicates a detached path, i.e., do not back propagate through this path.

2.2 Sequence to Sequence Generation

Sequence to Sequence (Seq2Seq) learning was first introduced by Sutskever et al. (2014); together with the soft attention mechanism of Bahdanau et al. (2014), it has been widely used in natural language generation tasks. Gülçehre et al. (2016); Gu et al. (2016) used a mixture of generation and pointing to overcome the problem of large vocabulary size. Paulus et al. (2017); Zhou et al. (2017) applied Seq2Seq models on summary generation tasks, while Du et al. (2017); Yuan et al. (2017) generated questions conditioned on documents and answers from machine comprehension datasets. Seq2Seq was also applied on neural sentence simplification (Zhang and Lapata, 2017) and paraphrase generation tasks (Xu et al., 2018).

2.3 Representation Learning for Language

Representation learning for language has been studied widely in the past few years. Mikolov et al. (2013) propose Word2Vec, in which a contrastive loss is used to predict context words given a focus word. Kiros et al. (2015) further propose Skip-thought vectors, which uses Recurrent Neural Networks to predict context sentences. Subramanian et al. (2018) leverage multi-task learning by sharing a single recurrent sentence encoder across weakly related tasks to learn general sentence representations. Logeswaran and Lee (2018) formulated the sentence-representation-learning task as a classification problem, where the classifier learns to distinguish a context sentence from contrastive negative samples based on their vector representations. Recently, van den Oord et al. (2018) proposed Contrastive Predicting Coding (CPC), which learns sentence representations by maximizing the mutual information between sequence encodings at different time-steps, also using a contrastive loss.

3 Model Architecture

Given a piece of source text, our objective is to generate a variable number of multi-word phrases. To this end, we opt for the sequence-to-sequence framework as the basis of our model, combined with attention and pointer softmax mechanisms in the decoder. To teach the model to vary the number of generated phrases, we join a variable number of multi-word phrases, separated by delimiters, as a single sequence. This concatenated sequence is then the target for sequential generation during training. An overview of our model's structure is shown in Figure 1.[1]

Notations

In the following subsections, we use w to denote input text tokens, x to denote token embeddings, h to denote hidden states, and y to denote output text tokens. Superscripts denote time-steps in a sequence, and subscripts e and d indicate whether a variable resides in the encoder or the decoder of

[1] We plan to release the datasets and code in the near future.

the model, respectively. The absence of a superscript indicates multiplicity in the time dimension. L refers to a linear transformation and L^f refers to it followed by a non-linear activation function f. Angled brackets, $\langle\rangle$, denote concatenation.

3.1 Source Encoding

Given a source text consisting of N words $w_e^1, \ldots, w_e^N$, the encoder converts these discrete symbols into a set of N real-valued vectors $h_e = (h_e^1, \ldots, h_e^N)$. Specifically, we first embed each word w^t into a embedding vector x_e^t, which is then fed into a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) for deriving h_e^t from contextual information in the source text:

$$\begin{aligned} h_{e,\text{fwd}}^t &= \text{LSTM}_{e,\text{fwd}}(x_e^t, h_{e,\text{fwd}}^{t-1}), \\ h_{e,\text{bwd}}^t &= \text{LSTM}_{e,\text{bwd}}(x_e^t, h_{e,\text{bwd}}^{t+1}), \\ h_e^t &= \langle h_{e,\text{fwd}}^t, h_{e,\text{bwd}}^t \rangle. \end{aligned} \tag{1}$$

Dropout (Srivastava et al., 2014) is applied to both x_e and h_e for regularization.

3.2 Attentive Decoding

The decoder is a recurrent model that takes the source encodings h_e and generates a distribution $p(y^t)$ over possible output tokens at each time-step t. With pointer softmax (Gülçehre et al., 2016), the target distribution consists of two parts: a distribution p_a over a prescribed vocabulary (abstractive), and a pointing distribution p_e over the tokens in the source text (extractive). We will focus on the derivation of p_a in this subsection.

The first component of the decoder is a uni-directional LSTM. At each time-step t, the decoding LSTM_d generates a new state h_d^t from the embedding vector x_d^t and its recurrent state h_d^{t-1}. Specifically, x_d^t is the embedding of w_d^t. During training by teacher forcing, w_d^t is the groundtruth target token at previous time-step $t-1$; during evaluation, $w_d^t = y^{t-1}$, is the prediction at the previous time-step,

$$h_d^t = \text{LSTM}_d(x_d^t, h_d^{t-1}). \tag{2}$$

The initial state h_d^0 is derived from the final encoder state h_e^N by applying a single-layer feed-forward neural net (FNN): $h_d^0 = L_0^{\tanh}(h_e^N)$. Dropout is applied to both the embeddings x_d and the LSTM states h_d.

In order to better incorporate information from the source text, an attention mechanism (Bahdanau et al., 2014) is employed when generating token y^t. The objective is to infer a notion of importance $\alpha^{t,i}$ for each source word w_e^i based on the current decoder state h_d^t. This is achieved by measuring the "energy" between h_d^t and the i-th source encoding h_e^i with a 2-layer FNN:

$$\text{energy}(h_d^t, h_e^i) = L_1(L_2^{\tanh}(\langle h_d^t, h_e^i \rangle)). \tag{3}$$

The output of the second layer is a scalar-valued energy score. The energies over all encoder states h_e thus define a distribution over the source sequence for each decoding step t:

$$\alpha^t = \text{softmax}(\text{energy}(h_d^t, h_e)). \tag{4}$$

To generate the new token y^t, the final step is to derive the generative distribution p_a by applying a 2-layer FNN to the concatenation of the decoder LSTM state h_d^t and the weighted sum of the source encodings weighted by the distribution α^t:

$$p_a(y^t) = L_3^{\text{softmax}}(L_4^{\tanh}(\langle h_d^t, \sum_i \alpha^{t,i} \cdot h_e^i \rangle)), \tag{5}$$

where the output size of the second layer equals to the target vocabulary size.

3.3 Pointer Softmax

We employ the pointer softmax (Gülçehre et al., 2016) mechanism to choose between generating tokens from the general vocabulary and pointing to tokens in the source text. Essentially, the pointer

softmax module computes a scalar switch s^t at each generation time-step and uses it to interpolate the abstractive distribution p_a over the vocabulary (see Equation 5) and the extractive distribution p_e over the source text tokens:

$$p(y^t) = s^t \cdot p_a(y^t) + (1 - s^t) \cdot p_e(y^t). \tag{6}$$

Semantically, the switch should be conditioned on both the source representation and the decoder state at each decoding step t. Specifically, we project the attention-weighted sum of the source encodings $\sum_i \alpha^{t,i} \cdot h_e^i$ and the decoder state h_d^t into the same space by two separate linear transformations, and further transform the sum of the resulting vectors into the scalar switch value s^t with a 1-layer FNN:

$$s^t = L_5^{\text{sigmoid}}(\tanh(L_6(\sum_i \alpha^{t,i} \cdot h_e^i) + L_7(h_d^t))). \tag{7}$$

We use the source attention weights α^t from Equation 4 as the extractive distribution for time-step t:

$$p_e(y^t) = \alpha^t. \tag{8}$$

3.4 Decoding Strategies

With the probability $p(y^t)$ defined in Equation 6, various decoding methods can be applied to decode the target word y^t. Selecting a decoding strategy is important because it determines whether the generated keyphrase set is of fixed or variable size.

To our best knowledge, all existing models generate fixed size sets by first over-generating a large number of candidate keyphrases, followed by some ranking algorithm to truncate the candidate set to a fixed number of final results. One major limitation is that such approaches are incompatible with the variable-number nature of keyphrases. In the KP20K dataset, for example, the average number of keyphrases in the training set is 5.27, while the variance is as high as 14.22. In addition, over-generation is usually achieved by setting a large beam size in beam search (e.g., 150 and 200 in Chen et al. (2018b); Meng et al. (2017), respectively), which is computationally rather expensive.

Since our proposed model is trained to generate a dynamic number of phrases as a single sequence joined by delimiters, $\langle$SEP$\rangle$, we can obtain variable length output by simply decoding a single sequence for each sample, either by greedy search or by taking the top beam sequence from beam search. The resulting model thus undertakes the additional task of dynamically estimating the proper size of the target phrase set. We will show in later sections that the model remains empirically competitive when compared to baselines that lack this desirable capacity. Another notable attribute of our decoding strategy is that, by generating a set of phrases in a single sequence, the model conditions its generation on all its generation history. Compared to the strategy used in previous works (i.e., phrases are generated in parallel by beam search, without interdependency), our model can learn dependencies among target phrases in a more explicit way.

3.5 Diversified Generation

There are typically multiple keyphrases for a given text because each keyphrase represents certain aspects of the text. Therefore keyphrase diversity is desired for the keyphrase generation, to increase the semantics coverage of source text and meanwhile to reduce redundancy in generated phrases. Most previous keyphrase generation models generate multiple phrases by over-generation, which is highly prone to generate similar phrases due to the nature of beam search. Given our objective to generate variable numbers of keyphrases, we need to adopt new strategies for achieving better diversity in the output.

Recall that we represent variable numbers of keyphrases as delimiter-separated sequences. One particular issue we observed during error analysis is that the model tends to produce identical tokens following the delimiter token. For example, suppose a target sequence contains n delimiter tokens at time-steps $t_1, \ldots, t_n$, respectively. During training, the model is rewarded for generating the same delimiter token at these time-steps, which presumably introduces much homogeneity in the corresponding decoder states $h_d^{t_1}, \ldots, h_d^{t_n}$. When these states are subsequently used as inputs at the time-steps immediately following the delimiter, the decoder naturally produces highly similar distributions over the following tokens, resulting in identical tokens being decoded. To alleviate this problem, we propose two plug-in components for the sequential generation model.

3.5.1 Orthogonal Regularization

We first propose to explicitly encourage the delimiter-generating decoder states to be different from each other. This is inspired by Bousmalis et al. (2016), who use orthogonal regularization to encourage representations across domains to be as distinct as possible. Specifically, we stack the decoder hidden states corresponding to delimiters together to form matrix $H = \langle h_d^{t_1}, \ldots, h_d^{t_n} \rangle$ and use the following equation as the orthogonal regularization loss:

$$\mathcal{L}_{\mathrm{OR}} = \left\| H^\top H - I_n \right\|_2, \tag{9}$$

where $H^\top$ is the matrix transpose of H, I_n is the identity matrix of rank n, $\|M\|_2$ indicates L^2 norm of a matrix M. This loss function prefers orthogonality among the hidden states $h_d^{t_1}, \ldots, h_d^{t_n}$ and thus improves diversity in the tokens following the delimiters.

3.5.2 Target Encoding

We propose an additional mechanism that focuses more on the semantic representations of generated phrases. Again, our goal here is to reduce covariance between certain decoder states and the corresponding target tokens (namely, delimiters). Specifically, we introduce another uni-directional recurrent model $\mathrm{LSTM}_{\mathrm{TE}}$, dubbed *target encoder*, which encodes decoder-generated tokens y^τ, where $\tau \in [0, t)$, into hidden states h_{TE}^t. This state is then taken as an extra input to the decoder LSTM, modifying Equation 2 to:

$$h_d^t = \mathrm{LSTM}_d(\langle x_d^t, h_{\mathrm{TE}}^t \rangle, h_d^{t-1}). \tag{10}$$

We do not train the target encoder with the training signal from generation (i.e., errors do not backpropagate from the decoder LSTM to the target encoder, as shown in Figure 1), because the resulting decoder would be equivalent to a 2-layer LSTM with residual connections. Instead, inspired by Logeswaran and Lee (2018); van den Oord et al. (2018), we train the target encoder in an unsupervised fashion. We extract target encoder hidden states h_{TE}^t for which w_d^t is the delimiter token, $\langle \texttt{SEP} \rangle$, and use these as phrase representations. We train by maximizing the mutual information between each of these phrase representations and the final state of the source encoder, h_e^T, as follows. For each phrase representation vector h_{TE}^t, we take a set $H_e^T = \{h_{e,1}^T, \ldots, h_{e,N}^T\}$ of N encodings of N different sources. This set contains one positive sample $h_{e,true}^T$, the encoder representation for the source sequence whose keyphrases are being generated, and $N-1$ negative samples from other source sequences (sampled at random from the training data). The target encoder is optimized on the classification loss,

$$\begin{aligned} \mathcal{L}_{\mathrm{TE}} &= -\log \frac{g(h_{e,true}^T, h_{\mathrm{TE}}^t)}{\sum_{i \in [1,N]} g(h_{e,i}^T, h_{\mathrm{TE}}^t)}, \\ g(h_a, h_b) &= \exp(h_a^\top B h_b), \end{aligned} \tag{11}$$

where B is bi-linear transformation.

The motivation here is to constrain the representation of each generated keyphrase to be semantically close to the overall meaning of the source text. With a recurrent architecture in the target encoder, the model can potentially help to capture phrase-level semantics in the generated output, and thus effectively address the diversity issue stemming from the delimiter tokens.

3.5.3 Training Loss

We adopt the widely used negative log-likelihood loss in our sequence generation model, denoted as $\mathcal{L}_{\mathrm{NLL}}$. The overall loss we use in our model is

$$\mathcal{L} = \mathcal{L}_{\mathrm{NLL}} + \lambda_{\mathrm{OR}} \cdot \mathcal{L}_{\mathrm{OR}} + \lambda_{\mathrm{TE}} \cdot \mathcal{L}_{\mathrm{TE}}, \tag{12}$$

where λ_{OR} and λ_{TE} are scalars. As shown in Figure 1, at each time-step t, the decoder LSTM has 3 inputs: embedding vector x_d^t of input token w_d^t, target encoding h_{TE}^t, and hidden state h_d^{t-1} from the previous time-step. Both h_{TE}^t and h_d^{t-1} are designed to boost generation diversity.

4 Datasets and Experiments

In this section we introduce the datasets we use for evaluation and report results from our models. In the following subsections, catSeq refers to the sequence-to-concatenated-sequences model described in Section 3, without the target encoder and orthogonal regularization; catSeqD refers to the model augmented with the target encoder and orthogonal regularization. During teacher forcing, to build the groundtruth sequences, we sort target keyphrases by their order of first occurrence in the source text, and append absent keyphrases at the end. This order may guide the attention mechanism to attend to source positions in a smoother way. Implementation details can be found in Appendix A.

4.1 Evaluation Methods

As mentioned in Section 3.4, previous models are only able to generate a fixed number of keyphrases, and oftentimes, they over-generate. As a result, these studies opt for evaluating either the top 5 or top 10 results against the groundtruth labels. We argue that this evaluation method is unsuitable for the keyphrase generation task and variable-size set generation tasks in general. To validate this hypothesis, we calculated the performance upper bounds for $F_1@5$ and $F_1@10$ on the KP20K validation set. For this we assume an oracle that generates the groundtruth sets of keyphrases, and when the number of generated keyphrases differs from 5 or 10, we insert random wrong answers. The $F_1@5$ and $F_1@10$ scores of this system are 0.858 and 0.626, respectively, suggesting that the existing evaluation approach is problematic. Furthermore, different datasets have different natures, so using the same arbitrarily chosen k to compute $F_1@k$ on all datasets may be inappropriate. Statistics of all datasets we use in this work are shown in Table 1.

Dataset	#Train	Valid	Test	Avg#	Var#
KP20K	567,830	20k	20k	5.27	14.22
INSPEC	–	1500	500	9.57	22.42
KRAPIVIN	–	1844	460	5.24	6.64
NUS	–	169	42	11.54	64.57
SEMEVAL	–	144	100	15.67	15.10
STACKEXCHANGE	298,965	16k	16k	2.69	1.37
TEXTWORLD ACG	12,837	575	575	9.96	25.01

Table 1: Statistics of datasets we use in this work. Avg# and Var# indicate the mean and variance of numbers of target phrases per data point, respectively.

On the other hand, a well-trained neural sequence generation model should have the ability to decide when to stop, i.e. decide how many keyphrases to predict by itself, by generating a stop token (e.g., $\langle$EOS$\rangle$). Taking all this into consideration, we propose two new evaluation methods for set generation tasks:

- $F_1@\mathcal{M}$ (number determined by model): For models that generate variable numbers of outputs, compute F_1 score between all phrases generated by the model with phrases in groundtruth.
- $F_1@\mathcal{V}$ (number determined by validation performance): For models that over-generate, take the k value that gives the highest $F_1@k$ on the validation set, report test performance using this k value. On data point i where the model still generates fewer than k outputs, $F_1@\mathcal{V}(i) = F_1@\mathcal{M}(i)$.

In the following subsections, we report our experimental results on multiple datasets and compare with CopyRNN[2] and other existing works.

During generation, we use two strategies: Greedy and Recall+. In the Greedy strategy, the decoder generates only one target sequence greedily, halting generation whenever an $\langle$EOS$\rangle$ token is generated. In the Recall+ strategy, we generate more outputs to boost recall by utilizing beam search, where each beam generates a sequence of keyphrases. We then follow Ye and Wang (2018)

[2]We compare all our results using new evaluation methods with CopyRNN, because by the time of writing, it's the only open sourced model enabling us to modify the evaluation code for fair comparisons.

to rank the generated set as follows: 1) Split each beam by ⟨SEP⟩, remove duplicates; 2) Beams with higher probabilities have higher rank; 3) Within a beam, keyphrases that are generated earlier have higher rank.

4.2 Experiments on Scientific Publication Datasets

Following (Meng et al., 2017; Chen et al., 2018a; Ye and Wang, 2018; Chen et al., 2018b), we use KP20K dataset to train our model and test it on a collection of scientific publication datasets, namely KP20K, INSPEC, KRAPIVIN, NUS, and SEMEVAL.

KP20K is a dataset constructed by (Meng et al., 2017) that comprises a large number of scientific publications. For each article, the abstract and title are used as the source text while the author keywords are used as target. The other four datasets contain many fewer articles, so we use them to test transferability of our model without tuning on them. Our proposed metric $F_1@\mathcal{V}$ requires validation data to determine the value of k, so for these four datasets, we use the training set for validation if it exists, and otherwise split the test set into two portions: we order all the data points by their file names alphabetically and take the first 20% as testing set and the rest 80% as validation set.

We report our model's performance on the *present*-keyphrase portion of the KP20K dataset in Table 2, using $F_1@\mathcal{M}$ and $F_1@\mathcal{V}$ metrics. To compare with previous works, we also compute $F_1@5$ and $F_1@10$ scores; when the model generates less than 5 or 10 keyphrases, we use whatever it generates to compare with groundtruth. From the table, we can see TG-Net performs the best on $F_1@5$, but the score drops immediately on the top 10 predictions, perhaps because the quality of 6_{th} to 10_{th} keyphrases are poor. An interesting observation is that, different from existing models, our model always has better scores on $F_1@10$ than $F_1@5$. This suggests that our model always has an accurate expectation regarding the number of keyphrases there should be, i.e., it decides to generate more than 5 keyphrases when necessary.

catSeqD outperforms catSeq on all metrics, suggesting the target encoder and orthogonal regularization help the model to generate higher quality keyphrases.

Model	$F_1@5$	$F_1@10$	$F_1@\mathcal{M}$	$F_1@\mathcal{V}$
Greedy				
catSeq	34.1	34.5	34.5	34.5
catSeqD	35.8	**36.1**	**36.2**	**36.2**
Recall+				
CopyRNN	32.8	25.5	–	33.1
Multi-Task	30.8	24.3	–	–
TG-Net	**37.2**	31.6	–	–
catSeq	33.8	34.5	34.6	34.6
catSeqD	35.0	35.8	35.9	35.9

Table 2: Present keyphrase predicting performance on KP20K test set. Compared with CopyRNN (Meng et al., 2017), Multi-Task (Ye and Wang, 2018), and TG-Net (Chen et al., 2018b).

We report our models' performance on the four transfer learning datasets in Table 3, using $F_1@\mathcal{M}$ and $F_1@\mathcal{V}$ for both Greedy and Recall+ modes. The scores are computed on the *present* portion of the four datasets. Our model performs competitively with CopyRNN on all four, and even outperforms it on NUS and SEMEVAL. From Table 1, we know the statistics of these four datasets are very different from KP20K. For example, in KP20K there are on average 5.27 keyphrases per data point, while SEMEVAL has 15.67. Without the ability to transfer, a model would memorize the distribution of the training data, thus having no chance to perform better on SEMEVAL than an over-generating model (e.g., CopyRNN). This is evidence that our model generalizes effectively on the scientific publication datasets.

On these datasets, generating *absent* keyphrases is extremely difficult and is far from solved. The current state-of-the-art model (Chen et al., 2018b) gets 0.267 and 0.075 recall on KP20K and SEMEVAL datasets, respectively, by forcing the model to generate 50 keyphrases. We argue that

Model	INSPEC	KRAPIVIN	NUS	SEMEVAL
Greedy ($F_1@\mathcal{M}$)				
catSeq	21.8	**32.0**	33.0	28.2
catSeqD	**22.5**	**32.0**	**35.9**	**30.6**
Recall+ ($F_1@\mathcal{V}$)				
CopyRNN	**34.8**	**32.4**	32.9	28.1
catSeq	28.3	28.3	35.9	28.9
catSeqD	30.9	27.4	**36.3**	**29.9**

Table 3: Model performance on transfer learning datasets.

evaluating a set generation task by only recall is problematic. In achieving higher recall by generating more keyphrases, a model sacrifices precision. An extreme case would be if a model generates all possible English phrases: recall would be high but precision would be close to 0. On the contrary, the precision score can be high if the model always generates only one keyphrase. According to our experiments, the F_1 scores on the *absent* portion of KP20K are around $1e^{-2}$, which makes it hard to compare different models' performance. We therefore do not report results on the absent portion and leave it for future work.

4.3 Experiments on STACKEXCHANGE Dataset

Model	Dev		Test	
	present	absent	present	absent
Greedy ($F_1@\mathcal{M}$)				
catSeq	53.7	9.6	54.5	9.2
catSeqD	**54.4**	**12.7**	**54.7**	**12.6**
Recall+ ($F_1@\mathcal{V}$)				
CopyRNN	57.6	24.6	58.0	24.8
catSeq	56.9	17.4	57.3	17.2
catSeqD	**61.5**	**26.2**	**61.1**	**26.6**

Table 4: Model performance on STACKEXCHANGE dataset.

Inspired by the StackLite tag recommendation task on Kaggle, we build a new benchmark based on the public StackExchange data.[3] From the raw data, we use question title and question text as the source, and use user-assigned tags as target keyphrases.

Since oftentimes the questions on StackExchange contain less information than in scientific publications, there are fewer keyphrases per data point in STACKEXCHANGE. Furthermore, StackExchange uses a tag recommendation system that suggests topic-relevant tags to users while submitting questions; therefore, we are more likely to see general terminology such as **`Linux`** and **`Java`**. This characteristic challenges models with respect to their ability to distill major topics of a question rather than selecting specific snippets from the text.

We report our models' performance on STACKEXCHANGE in Table 4. Results show catSeqD performs the best with both generation strategies; on the *absent*-keyphrase generation tasks, it outperforms catSeq by a large margin.

4.4 Text-based Game Command Generation

Text-based games are receiving more attention recently in the NLP, machine learning, and reinforcement learning communities. They are turn-based games in which all communications between game engine and player occur through text. At each game step, an agent receives a text observation

[3]https://archive.org/details/stackexchange, we choose 19 computer science related topics from Oct. 2017 dump.

	Dev	Test	Dev	Test
Model	Greedy ($F_1@\mathcal{M}$)		Recall+ ($F_1@\mathcal{V}$)	
CopyRNN	–	–	66.9	68.5
catSeq	88.3	85.4	87.2	**88.4**
catSeqD	90.6	**86.2**	87.3	88.1

Table 5: Model performance on TEXTWORLD ACG dataset.

Source	a visual test development environment for gui systems We have implemented an experimental test development environment (TDE) intended to raise the effectiveness of tests produced for GUI systems, and raise the productivity of the GUI system tester.The environment links a test designer, a test design library, and a test generation engine with a standard commercial capture/replay tool. These components provide a human tester the capabilities to capture sequences of interactions with the system under test (SUT), to visually manipulate and modify the sequences, and to create test designs that represent multiple individual test sequences. Test development is done using a high-level model of the SUT's GUI, and graphical representations of test designs. TDE performs certain test maintenance tasks automatically, permitting previously written test scripts to run on a revised version of the SUT.
catSeq	test development ; test development environment ; test ; test generation
catSeqD	test generation ; gui ; tool ; version ; capabilities ; systems ; design ; test ; human ; generation
Groundtruth	engine ; developer ; design ; human ; standardization ; tool ; links ; graphics ; model ; libraries ; replay ; component ; interaction ; product ; development environment ; script ; visualization ; capabilities ; systems ; experimentation ; test designer ; environments ; test generation ; testing ; maintenance ; test maintenance ; version ; effect ; sequence

Table 6: Example from KP20K validation set, predictions generated by catSeq and catSeqD models.

describing the environment as input, then issues a text command to take action in the game. The engine responds in turn with textual feedback and a numerical reward.

We here consider the *admissible* commands that an agent can issue in response to an observation to be keyphrases. Admissible commands are those that have some effect on the environment described by the text observation, and thus require picking out key details from it. In this formulation, the observation is the source text and the set of admissible commands forms the target.

To generate keyphrase-style training data from text-based games, we use TextWorld (Côté et al., 2018), a text-based learning environment, to procedurally generate a collection of games and their playthroughs. Some example (observation, command-set) datapoints are shown in Appendix C. We call this dataset TEXTWORLD ACG, its statistics can be found in Table 1.

We report our models' performance on this dataset in Table 5. Note that since almost all target commands are absent from the source text, we do not differentiate present and absent keyphrases in this dataset. From the table, we can see catSeqD outperforms catSeq with the Greedy generation strategy and gets similar performance when over-generating. Both of our models outperform CopyRNN by a large margin.

5 Analysis and Discussion

5.1 Effect of Target Encoding and Orthogonal Regularization

To verify our assumption that target encoding and orthogonal regularization help to boost the diversity of generated sequences, we use two metrics, one quantitative and one qualitative, to measure diversity of generation. First, we simply calculate the average unique predictions on both KP20K and TEXTWORLD ACG datasets produced by both catSeq and catSeqD. We observe that average unique predictions from catSeqD are consistently slightly greater than those from catSeq, and give further details in Appendix B.

Second, from the model running on the KP20K validation set, we randomly sample 2000 decoder hidden states at k steps following a delimiter ($k = 1, 2, 3$) and apply an unsupervised clustering method (t-SNE (van der Maaten and Hinton, 2008)) on them. From the Figure 2 we can see that hidden states sampled from catSeqD are easier to cluster while hidden states sampled from catSeq yield one mass of vectors with no obvious distinct clusters. Results on both metrics suggest target encoding and orthogonal regularization indeed help diversifying generation of our model.

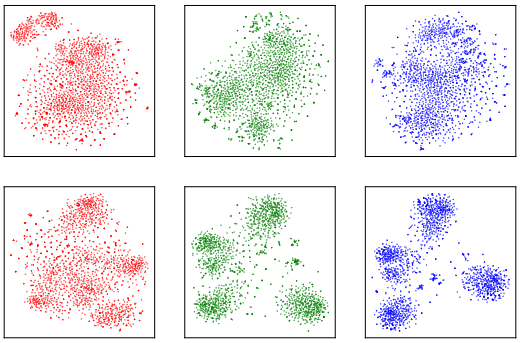

Figure 2: t-SNE results on decoder hidden states. Upper row: catSeq; lower row: catSeqD; column k shows hidden states sampled from tokens at k steps following a delimiter.

5.2 Result Examples

To illustrate the difference of predictions between catSeq and catSeqD, we show an example chosen from the KP20K validation set in Table 6. In this example there are 29 groundtruth phrases. Neither of the models is able to generate all of the keyphrases, but it is obvious that the predictions from catSeq all start with "test", while predictions from catSeqD are diverse. This to some extent verifies our assumption that without the target encoder and orthogonal regularization, decoder states generated from delimiter $\langle$SEP$\rangle$ are less diverse.

6 Conclusion and Future Work

We propose a recurrent generative model that generates multiple keyphrases sequentially, with two extra modules that promote generation diversity. We propose new metrics to evaluate keyphrase generation. Our model shows promising performance on three keyphrase generation datasets including 2 newly introduced in this work. In future work, we plan to investigate how target phrase order affects the generation behavior, and further explore ways to generate set in an order invariant way. We would also like to investigate how to leverage reinforcement learning to help keyphrase generation.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., and Erhan, D. (2016). Domain separation networks. *CoRR*, abs/1608.06019.

Chen, J., Zhang, X., Wu, Y., Yan, Z., and Li, Z. (2018a). Keyphrase generation with correlation constraints. *CoRR*, abs/1808.07185.

Chen, W., Gao, Y., Zhang, J., King, I., and Lyu, M. R. (2018b). Title-guided encoding for keyphrase generation. *CoRR*, abs/1808.08575.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., Asri, L. E., Adada, M., Tay, W., and Trischler, A. (2018). Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Du, X., Shao, J., and Cardie, C. (2017). Learning to ask: Neural question generation for reading comprehension. *CoRR*, abs/1705.00106.

Gollapalli, S. D., Li, X., and Yang, P. (2017). Incorporating expert knowledge into keyphrase extraction. In *AAAI*, pages 3180–3187. AAAI Press.

Gu, J., Lu, Z., Li, H., and Li, V. O. K. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *CoRR*, abs/1603.06393.

Gülçehre, Ç., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. *CoRR*, abs/1603.08148.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3294–3302. Curran Associates, Inc.

Le, T. T. N., Nguyen, M. L., and Shimazu, A. (2016). Unsupervised keyphrase extraction: Introducing new kinds of words to keyphrases. *29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016*.

Liu, Z., Chen, X., Zheng, Y., and Sun, M. (2011). Automatic keyphrase extraction by bridging vocabulary gap. *the Fifteenth Conference on Computational Natural Language Learning*.

Logeswaran, L. and Lee, H. (2018). An efficient framework for learning sentence representations. *CoRR*, abs/1803.02893.

Lopez, P. and Romary, L. (2010). Humb: Automatic key term extraction from scientific articles in grobidp. *the 5th International Workshop on Semantic Evaluation*.

Luan, Y., Ostendorf, M., and Hajishirzi, H. (2017). Scientific information extraction with semi-supervised neural tagging. *CoRR*, abs/1708.06075.

Medelyan, O., Frank, E., and Witten, I. H. (2009). Human-competitive tagging using automatic keyphrase extraction. *Empirical Methods in Natural Language*.

Meng, R., Zhao, S., Han, S., He, D., Brusilovsky, P., and Chi, Y. (2017). Deep keyphrase generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 582–592. Association for Computational Linguistics.

Mihalcea, R. and Tarau, P. (2004). Textrank: Bringing order into text. *Empirical Methods in Natural Language*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. (2018). Learning general purpose distributed sentence representations via large scale multi-task learning. *CoRR*, abs/1804.00079.

Subramanian, S., Wang, T., Yuan, X., and Trischler, A. (2017). Neural models for key phrase detection and question generation. *CoRR*, abs/1706.04560.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

Wan, X. and Xiao, J. (2008). Single document keyphrase extraction using neighborhood knowledge. *AAAI*.

Wang, M., Zhao, B., and Huang, Y. (2016). Ptr: Phrase-based topical ranking for automatic keyphrase extraction in scientific publications. *23rd International Conference, ICONIP 2016*.

Witten, I. H., Paynter, G. W., Frank, E., Gutwin, C., and Nevill-Manning, C. G. (1999). Kea: Practical automatic keyphrase extraction. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, DL '99, pages 254–255, New York, NY, USA. ACM.

Xu, Q., Zhang, J., Qu, L., Xie, L., and Nock, R. (2018). D-page: Diverse paraphrase generation. *CoRR*.

Yang, Z., Hu, J., Salakhutdinov, R., and Cohen, W. W. (2017). Semi-supervised qa with generative domain-adaptive nets. In *the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Ye, H. and Wang, L. (2018). Semi-supervised learning for neural keyphrase generation. *CoRR*, abs/1808.06773.

Yuan, X., Wang, T., Gülçehre, Ç., Sordoni, A., Bachman, P., Subramanian, S., Zhang, S., and Trischler, A. (2017). Machine comprehension by text-to-text neural question generation. *CoRR*, abs/1705.02012.

Zhang, Q., Wang, Y., Gong, Y., and Huang, X. (2016). Keyphrase extraction using deep recurrent neural networks on twitter. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 836–845. Association for Computational Linguistics.

Zhang, X. and Lapata, M. (2017). Sentence simplification with deep reinforcement learning. *CoRR*, abs/1703.10931.

Zhou, Q., Yang, N., Wei, F., and Zhou, M. (2017). Selective encoding for abstractive sentence summarization. *CoRR*, abs/1704.07073.

A Implementation Details

Implemntation details of our proposed models are as follows. In all experiments, the word embeddings are initialized with 150-dimensional random matrices. The number of hidden units in both the encoder and decoder LSTM are 300. The number of hidden units in target encoder LSTM is 64. The size of vocabulary is 50,000.

The numbers of hidden units in MLPs described in Section 3 are as follows.

MLP	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
Dimension	300	300	1	50k	300	1	64	64

During target encoder training, we maintain a queue of size 128, to store the recent source representations. During negative sampling, we randomly sample 32 samples from this queue, thus target encoding loss in Equation 11 is a 33-way classification loss. In catSeqD, we set both the λ_{OR} and λ_{TE} in Equation 12 to be 0.03. In all experiments, we use a dropout rate of 0.3.

We use *adam* (Kingma and Ba, 2014) as the step rule for optimization. The learning rate is $1e^{-3}$. The model is implemented using *PyTorch* (Paszke et al., 2017).

B Average Amount of Unique Predictions by Models

	catSeq		catSeqD	
Dataset	Greedy	Recall+	Greedy	Recall+
KP20K	3.43	5.05	3.54	5.23
TEXTWORLD ACG	8.24	10.97	8.88	11.30

Table 7: Average number of generated keyphrase on KP20K and TEXTWORLD ACG.

C TEXTWORLD ACG Data Examples

See Table 8

Observations	-= Kitchen =- You find yourself in a kitchen. An usual one. Let's see what's in here. You can see a closed refrigerator in the room. You see a counter. Why don't you take a picture of it, it'll last longer! The counter appears to be empty. You swear loudly. You can see a kitchen island. You shudder, but continue examining the kitchen island. The kitchen island is typical. On the kitchen island you can see a note. You can see a stove. On the stove you make out a tomato plant. You idly wonder how they came up with the name TextWorld for this place. It's pretty fitting. There is an open screen door leading east. There is an open wooden door leading west. There is an unblocked exit to the north. You need an unblocked exit? You should try going south. You put the tomato plant on the stove. You are carrying: an old key.
Admissible Commands	close screen door; close wooden door; cook tomato plant; drop old key; open refrigerator; go east; go north; go south; go west; put old key on counter; put old key on kitchen island; put old key on stove; take note from kitchen island; take tomato plant from stove
Observations	-= Bedroom =- You're now in a bedroom. You make out a chest drawer. Look over there! an antique trunk. The antique trunk contains an old key. You make out a king-size bed. But there isn't a thing on it. What, you think everything in TextWorld should have stuff on it? There is a closed wooden door leading east. You open the antique trunk, revealing an old key. You are carrying nothing.
Admissible Commands	close antique trunk ; open chest drawer ; take old key from antique trunk

Table 8: Example observations and admissible commands in TEXTWORLD ACG dataset.