ENCONTER: Entity Constrained Progressive Sequence Generation via Insertion-based Transformer

Lee-Hsun Hsieh **Yang-Yin Lee** **Ee-Peng Lim**
Singapore Management University
Singapore
jesus255221@gmail.com yylee@smu.edu.sg eplim@smu.edu.sg

Abstract

Pretrained using large amount of data, autoregressive language models are able to generate high quality sequences. However, these models do not perform well under hard lexical constraints as they lack fine control of content generation process. Progressive insertion-based transformers can overcome the above limitation and efficiently generate a sequence in parallel given some input tokens as constraint. These transformers however may fail to support hard lexical constraints as their generation process is more likely to terminate prematurely. The paper analyses such early termination problems and proposes the ENtity-CONstrained insertion TransformER (ENCONTER), a new insertion transformer that addresses the above pitfall without compromising much generation efficiency. We introduce a new training strategy that considers predefined hard lexical constraints (e.g., entities to be included in the generated sequence). Our experiments show that ENCONTER outperforms other baseline models in several performance metrics rendering it more suitable in practical applications. [1]

1 Introduction

The field of Natural Language Generation (NLG) (Gatt and Krahmer, 2018) has seen significant improvements in recent years across many applications such as neural machine translation (Bahdanau et al., 2015), text summarization (Chopra et al., 2016), poem generation (Zugarini et al., 2019) and recipe generation (H. Lee et al., 2020). Constrained text generation (CTG) is one of the challenging problems in NLG that is important to many real world applications but has not been well addressed. CTG imposes input constraints which may be in the form of objects expected to exist in the generated text or rules over objects in the generated text (Hokamp and Liu, 2017). The objects here can be entities, phrases, predefined nouns, verbs, or sentence fragments. The constraints can be categorized into two types: (1) *Hard-constraints* which require mandatory inclusion of certain objects and complete compliance of given rules (Post and Vilar, 2018; Hu et al., 2019; Miao et al., 2019; Welleck et al., 2019; Zhang et al., 2020); and (2) *Soft-constraints* which allow the some constraint objects or rules to be not strictly enforced in the generated text (Qin et al., 2019; Tang et al., 2019). As autoregressive models generate tokens from left to right, they cannot easily support constraints involving multiple input objects, hard-constrained text generation therefore often requires non-autoregressive models.

Recently, Zhang et al. (2020) proposed a non-autoregressive hard-constrained text generation model (POINTER) that generates a text sequence in a progressive manner using an insertion-transformer (Stern et al., 2019). To train an insertion transformer to generate a missing token between every two tokens in an input sequence, the training data is prepared by masking "less important" tokens in the original text sequence in an alternating manner. The process is then repeated using the masked input sequence as the new original sequence, and further masking alternate tokens in it. The process ends when the masked sequence meets some length criteria.

While POINTER shows promising results, it does not consider hard constraints which involve entities that must be included in the generated sequence. Such entity constraint requirements are unfortunately prevalent in many applications. For example, we may want to generate a job description with some given skills, or a food recipe with some given ingredients.

A naive approach to the problem is to apply con-

[1]Our code is available at https://github.com/LARC-CMU-SMU/Enconter

straints on the POINTER's masking strategy forcing it to keep entity tokens. We call this modified model POINTER-E. Although this allow entity information entering POINTER-E, another problem rises. POINTER-E suffers from *cold start problem* which refers to the inability to generate meaningful tokens at the early stages of inference forcing the generation to end prematurely. This issue can be attributed to the POINTER-E's top-down masking strategy for training the insertion transformer and the tokens of input entities not evenly spread out across the sequence.

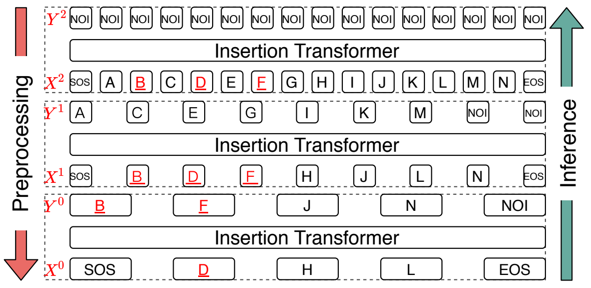
To solve the cold start generation problem, we propose ENCONTER that incorporates bottom-up masking strategy. ENCONTER supports hard entity constraints, and encourages more meaningful tokens to be generated in the early stages of generation thus reducing cold start. On top of that, we further introduce the balanced binary tree scheme (Stern et al., 2019) to reduce the number of stages in generation and to improve the efficiency of generation.

2 Entity Constrained Sequence Generation

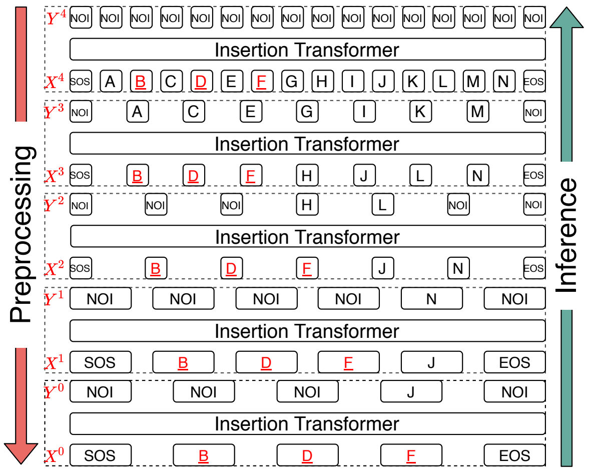
In this section, we first describe the state-of-the-art POINTER model, its preprocessing of training data and inference process. We highlight the pitfalls of the entity constrained variant of POINTER, POINTER-E. We then present our proposed entity constrained insertion transformer called ENCONTER.

2.1 POINTER

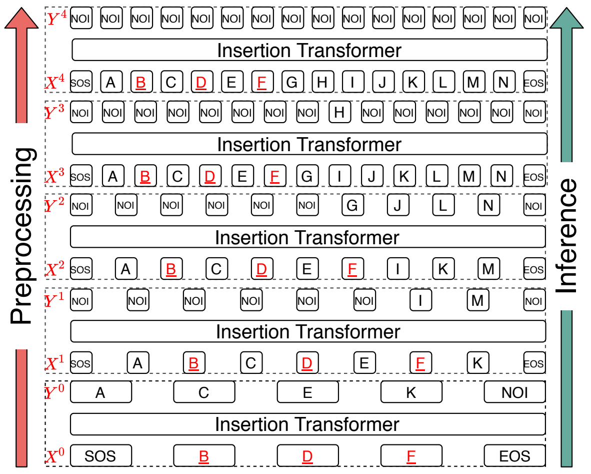
POINTER adopts a progressive masking approach to train an insertion transformer. Let $X = \{x_1, x_2, \dots, x_T\}$ denote a a sequence where $x_t \in V$, where T is the sequence length and V is a finite vocabulary set. Suppose X is a training sequence, POINTER preprocesses it to obtain the training pairs $S = \{(X^k, Y^k) | k \in \{K, \dots, 0\}\}$ using a progressive masking strategy. As shown in Figure 1a, in each stage X^k represents the input sequence for stage k, and Y^k represents the sequence of masked tokens to be inferred. X^K is identical to the final training sequence $X^K = X$, and there should not be any additional tokens to infer. X^0 on the other hand represents the initial lexical constraints. In stage k, Y^k are the tokens to be predicted between adjacent tokens of X^k. A special no-insertion token $[NOI]$ is added to the vocabulary V and used



(a) POINTER masking.



(b) POINTER-E masking.



(c) ENCONTER insertion.

Figure 1: POINTER, POINTER-E, and ENCONTER with original sequence $X = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N\}$ where B,D, and F are the tokens forming the entity constraints. The stopping criteria for POINTER is set to $n = 3$.

in Y^k to indicate that no token is to be generated between adjacent tokens. Y^K is thus a sequence of all $[NOI]$'s indicating the end of generation. WordPiece (Wu et al., 2016) tokenization is applied in POINTER, and tokens split from the same word share the same score.

Token importance scoring POINTER assigns each token $x_t \in X$ an importance score α_t:

$$\alpha_t = \alpha_t^{TF-IDF} + \alpha_t^{POS} + \alpha_t^{YAKE}, \quad (1)$$

where α_t^{TF-IDF}, α_t^{POS}, and α_t^{YAKE} denote term frequency-inverse document frequency (TF-IDF), POS tag scores and YAKE (Campos et al., 2020) keyword scores, respectively. These scores are normalized to [0,1]. α_t^{POS} is defined such that the scores of nouns and verbs are higher than those of other POS tags. The token importance scores are used to derive the masking pattern Y^{k-1} of stage $k-1$ from X^k.

POINTER adopts four criteria to derive Y^{k-1} from X^k: (1) Y^{k-1} can only include non-adjacent tokens in X^k; (2) the number of tokens to be masked are maximized in each stage to make the model more efficient; (3) less important tokens are masked before more important ones and (4) A stopping criteria n is defined. The algorithm stops when $|X^k| = n$. Kadane's algorithm (Gries, 1982) has been use in POINTER to fulfill the criteria. Specifically, the algorithm selects as many unimportant tokens as possible to be masked while not masking two adjacent tokens. X^0 is automatically determined when $|X^k| = n$, it does not necessarily match the way the initial input sequence is provided by real world applications or users, including the entity constraints.

Inference Given X^0 as input sequence, POINTER starts to infer $\hat{Y}^0$ and combines the two sequences to get $\hat{X}^1 = \left\{\hat{x}_1^0, \hat{y}_1^0, \hat{x}_2^0, \hat{y}_2^0, \ldots, \hat{x}_{|\hat{X}^0|}^0, \hat{y}_{|\hat{X}^0|}^0\right\}$. If $\hat{y}_t^0$ happens to be $[NOI]$, it will be deleted and leaving only non-$[NOI]$ tokens in $\hat{X}^1$. The process repeats until all the generated tokens in $\hat{Y}^k$ are $[NOI]$s.

As shown in Figure 1a, entities may not be preserved during the preprocessing steps and the lexical constraint X^0 is not guaranteed to cover entity constraint X^e even entity tokens are assigned high importance scores. The trained POINTER therefore may not be able generate a sequence successfully when given entity constraints during the inference. We therefore propose some changes to POINTER to make it entity-aware.

2.2 Entity Aware POINTER (POINTER-E)

The entity-aware POINTER model, POINTER-E, adopts a different preprocessing approach. Let $X^e \subset X$ be an ordered sequence of entity tokens (e.g., the person names in a news document). As X^e is likely to be used as the initial generation input (i.e., $X^0 = X^e$), POINTER-E's preprocessing does not mask these entity tokens over the different preprocessing stages. This way, the model is trained to focus on generating tokens around the entities. Such tokens form the context around the entities and context relating one entity to others. We achieve such goal by ignoring the importance scores applied on entity tokens. That is, we only compute α_t for $x_t \notin X^e$.

We then apply the POINTER's masking strategy on the sub-sequence between every two entity tokens in X. Suppose $(x_i, x_j) \subset X$ is a subsequence spanned by two entity tokens $\{x_l^e = x_i, x_{l+1}^e = x_j\} \in X^e$ where $l \in \{0, \ldots, |X^e| - 1\}$. Masking is applied on this subsequence iteratively until only $\{x_l^e, x_{l+1}^e\}$ are left:

$$\begin{aligned} S = &\{(X^K = X, Y^K), \ldots, \\ &(X^0 = X^e, Y^0)\}. \end{aligned} \quad (2)$$

As shown in Figure 1b, POINTER-E always picks the optimal masking patterns while preserving the entities.

Cold Start Problem While POINTER-E is aware of entities, entities in X^e may appear very close or very far from one another in the full sequence X, i.e., the gap between entities in the sequence X can vary a lot. Consider two sub-sequences $(x_i = x_l^e, x_j = x_{l+1}^e), (x_u = x_w^e, x_v = x_{w+1}^e) \subset X$ where $w, l \in (0, T_e - 1)$ and $w \neq l$. Suppose $j - i \gg v - u$. The tokens between (x_u, x_v) will then be masked out long before tokens in (x_i, x_j) during preprocessing and training. This results in POINTER-E trained to generate a lot of $[NOI]$s in Y^k for small k's. Figure 1b depicts this cold start problem as entity tokens B, D and E are near one another in X. As tokens between them are masked in early stages, the masked sequences in stages 0 and 1, Y^0 and Y^1, contain many $[NOI]$ tokens. POINTER-E trained with such data will therefore lack the ability to generate meaningful tokens in-between these entity tokens. In the worst case, POINTER-E simply generates all $[NOI]$ tokens and ends the generation prematurely which is known as the *cold start problem*.

To better show the problem, we define:

$$NOI\ ratio = \frac{\#[NOI] \text{ in } Y^k}{\#tokens \text{ in } Y^k}. \quad (3)$$

A clear problem of high $NOI\ ratio$ is that Y^k is very similar to Y^{k+1}. When $NOI\ ratio = 1$, the generation will end, In cases where $NOI\ ratio$ is very high for masked sequences in early stages, say Y^0, the trained POINTER-E will more likely infer from X^0 all $[NOI]$'s for $\hat{Y}^0$ and end the generation process. To address this, we need to re-examine the top-down masking stratey used in POINTER and POINTER-E.

2.3 ENCONTER

In this section, we propose ENCONTER which adopts a bottom-up masking strategy to overcome the cold start problem. There are two variants: GREEDY ENCONTER and BBT-ENCONTER.

GREEDY ENCONTER Different from POINTER-E, we now construct training pairs S from X by setting X^0 to be X^e:

$$\begin{aligned} S =&\{(X^0 = X^e, Y^0), (X^1, Y^1), \dots, \\ &(X^K = X, Y^K)\}, \end{aligned} \tag{4}$$

where Y^k represents the sequence of masked tokens to be inserted into X^k to form X^{k+1}. Similar to POINTER, Y^K contains $[NOI]$'s only. For every two adjacent tokens $\{x_t^k, x_{t+1}^k\} \in X^k$ where $t \in \{0, \dots, |X^k| - 1\}$, we insert a mask token. Let $\{x_t^k = x_i, x_{t+1}^k = x_j\}$ and (x_i, x_j) be the span of (x_i, x_j) in X. If $i + 1 = j$, the mask token is $[NOI]$. Otherwise, we select a token $x_{t'}$ from (x_i, x_j) with maximum importance score $\alpha_{t'}$ within (x_i, x_j) as the mask token. The sequence Y^k is formed after we go through all the t's. By inserting Y^k into X^k, we obtain the next sequence X^{k+1}. The iterative process stops when all the tokens to be inserted are $[NOI]$s. This method GREEDY ENCONTER greedily selects the token with maximum importance score in the span to be generated in a bottom up insertion (or unmasking) process. By forcing more non-$[NOI]$ tokens to be included in Y^0 and Y^k of small k's, *Greedy Encon-ter* achieves lower $NOI\ ratio$ in the early stages of inference. Experimentally, we find that the cold start problem is eliminated.

Balanced binary tree ENCONTER (BBT-ENCONTER) To further improve the efficiency of GREEDY ENCONTER, we incorporate balanced binary tree (Stern et al., 2019) into ENCONTER to bias the masking of tokens to be those near the center of the unobserved subsequence of tokens. BBT reward is added to the importance score function as follows. Suppose x_i and x_j are two adjacent tokens in X^k, and (x_i, x_j) represents the corresponding subsequence in X. We define the distance d_p for token $x_p \in (x_i, x_j)$ as:

$$d_p = min(p - i, j - p). \tag{5}$$

We use a softmax function to compute the reward for weighted score based on d_p:

$$w_p = \frac{exp(d_p/\tau)}{\sum_{k=i}^{j} exp(d_k/\tau)}. \tag{6}$$

The weights in the span are then normalized to $[0, 1]$. Then the importance score is defined as:

$$\alpha_p = w_p \cdot (\alpha_p^{TF-IDF} + \alpha_p^{POS} + \alpha_p^{YAKE}). \tag{7}$$

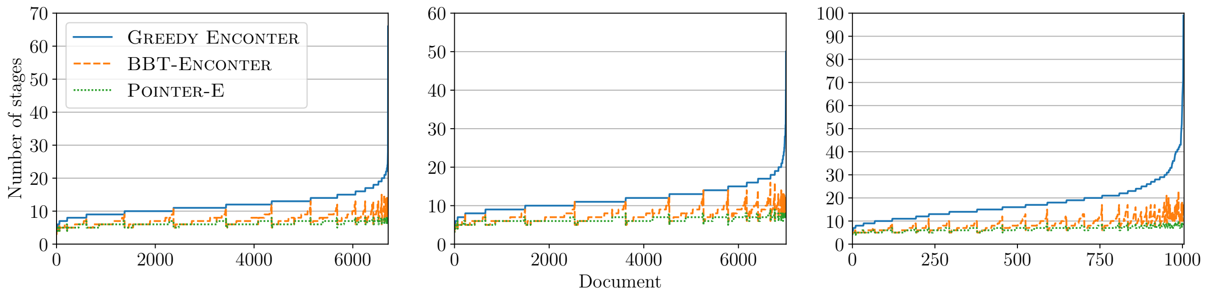
The construction of S is almost the same as GREEDY ENCONTER. The only difference is the new importance score function defined by Eq. 7. This proposed model, known as BBT-ENCONTER, will predict the center and semantically important token in X between two adjacent tokens of X^k.

2.4 Models with Entity Span Aware Inference Option (ESAI)

So far, all above-mentioned models assume that each entity consists of one single token. In real world use cases, an entity may contain more than one token. Without any control during the inference process, it is possible for other tokens to be generated in-between tokens of the same entity. For example in Table 5, "Group Consolidation" may be split into "handling Group s project / Consolidation". To avoid inserting any tokens in between any multi-token entity, we introduce the *entity span aware inference* option to the inference process of POINTER-E and ENCONTER to force the inference of $\hat{Y}^k$ to always generate $[NOI]$ in between the tokens of the multi-token entities. After applying ESAI, the multi-token entities will remain unbroken duing the generation process.

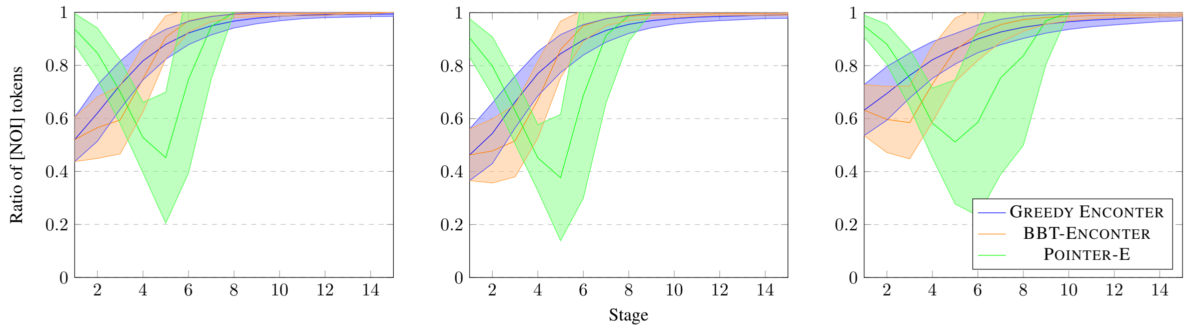
3 Empirical Analysis of POINTER-E and ENCONTER

In this section, we conduct an analysis of the data preprocessing step in POINTER-E, GREEDY ENCONTER and BBT-ENCONTER. Our objective is to empirically evaluate the characteristics of training data generated for the two models. We have left out POINTER as it is inherently not entity-aware and POINTER-E is its entity-aware variant. We first present the two datasets used in this study.



(a) SD (b) SM (c) CoNLL-2003

Figure 2: Number of stages of each document in SD, SM, and CoNLL-2003 (sorted).



(a) SD (b) SM (c) CoNLL-2003

Figure 3: Mean and standard deviation of the ratio of inserted/masked [NOI] tokens in each stage. All x axis are capped to 15 stages. The original maximum number of stages of (a), (b), and (c) are 67, 51, and 100, respectively.

3.1 Datasets

CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003): We select the English version which contains 1,393 news articles labeled with four named entity types: persons, locations, organizations and names of miscellaneous. Training and development sets are used to train the model. Documents having more than 512 tokens by wordpiece tokenizer used in BERT (Devlin et al., 2019) are discarded to ensure that the whole document can fit into the models.

Jobs: This is a job post dataset collected from Singapore's Jobsbank [2]. The dataset consists of 7,474 job posts under the software developer occupation (SD) and 7,768 job posts under the sales and marketing manager occupation (SM). We extract the requirement section of these job posts as the text sequences to be generated. For each requirement text sequence (or document), we use a dictionary of skills to annotate the skill and job related entities in the sequence.

The detailed information of the datasets can be found in Table 1. Table 1 reveals that Pointer-E has much higher $NOI\ ratio$ than Enconter in all the datasets.

[2]https://www.mycareersfuture.sg/

	CoNLL	**SM**	**SD**
#training docs	1,004	6,715	7,006
#testing docs	231	754	761
Avg length	220.7	99.4	121.1
Avg entities	24.6	24.4	27.7
#training pairs			
Pointer-E	6,557	43,913	41,343
Greedy Enconter	17,694	83,587	79,467
BBT-Enconter	8,492	52,609	48,625
$NOI\ ratio$ of Y^0			
Pointer-E	0.820	0.904	0.936
Greedy Enconter	0.546	0.463	0.519
BBT-Enconter	0.546	0.463	0.519

Table 1: Summary of the datasets. #training pairs refers to the total number of training pairs derive from each dataset

3.2 Analysis of NOI ratio and Stage Counts

We first analyse the ratio of $[NOI]$ tokens inserted or masked in every stage of the *training data*. Figure 3 shows the mean together with one standard deviation of the Pointer-E, Greedy Enconter and BBT-Enconter for each dataset. X-axis is in log scale. Note we add 1 to the stage number for showing log scale (e.g., the 10^0 in the figure indicates the ratio of $[NOI]$ tokens in Y^0). From Figure 3, we find all datasets share a few similar characteristics, namely: (1) for Pointer-E, the $[NOI]$ ratio is quite high in the first few stages, and drops when the stage is higher. A sudden increase of the ratio to 1 is due to the ending sequence consists all $[NOI]$'s; (2) for Enconter the $[NOI]$

ratio is low in the first few stages, and slowly increase to 1. The result shows ENCONTER can learn to generate balance proportion of $[NOI]$ and non-$[NOI]$ tokens in the first few stages, and also learn not to generate to many non-$[NOI]$ tokens when approaching the end of the generation process.

Figure 2 shows the number of stages each training document requires under different models. The numbers are sorted according to the following priority: GREEDY ENCONTER, POINTER-E, then BBT-ENCONTER. Since BBT-ENCONTER incorporates the binary tree reward scheme, it is able to perform insertion in the middle stages more efficiently compare to GREEDY ENCONTER. This helps to lower the total number of stages required to derive training pairs.

4 Experiment

4.1 Models for Comparison

GPT-2 (Radford et al., 2019) GPT-2 can be used to conduct conditional generation as well (soft-constraints). For a training sequence X together with its entities X^e, we concatenate X^e with X to form a training sequence $\{X^e, X\}$. X^e is then served as a control code sequence to guide GPT-2 in the generation of X. We fine-tune the *GPT-2 small* pretrained by huggingface [3] with 10^{-5} learning rate. Warmup and weight decay are applied. 10 epochs are used for fine-tuning.

POINTER-E, GREEDY ENCONTER, and BBT-ENCONTER: We use BERT (Devlin et al., 2019) as the underlying insertion transformer for all these models similar to that of POINTER. Specifically, we use the *bert-based-cased* pretrained by huggingface. BERT with language model head is fine-tuned on all the training pairs to obtain the models. Learning rate is set to 10^{-5} with warmup and weight decay. 10 epochs are used for fine-tuning.

For POINTER-E, GREEDY ENCONTER, and BBT-ENCONTER, top-k (top-20) sampling method is used to derive $\hat{Y}^k$. For GPT-2, we feed in the $\hat{X}^e$ and let GPT-2 generate the following tokens until reaching the end-of-generation token.

4.2 Evaluation Metrics

We evaluate the models using a few criteria, namely: recall of entities, quality with respect to human crafted text, diversity, fluency, cold start, and generation efficiency. We measure recall of entity constraints by the proportion of entity tokens found in the generated text. Even without ESAI, the recall metric will allow to compare the recall ability of models. Besides recall, we also consider BLEU (Papineni et al., 2002), METEOR (MTR) (Lavie and Agarwal, 2007) and NIST (Doddington, 2002), which are common metrics for evaluating the quality of generated text against human craft text. We compute the BLEU-2 (B-2) and BLEU-4 (B-4) which are n-gram precision-based metrics. For the BLUE based evaluation metric NIST, we compute the NIST-2 (N-2) and NIST-4 (N-4). To measure the diversity of generation, Entropy (Zhang et al., 2018) and Distinction (Li et al., 2016) are used. Entropy-4 (E-4) is defined as the frequency distribution of unique 4-gram terms. Dist-1 (D-1) and Dist-2 (D-2) are used to derive distinct n-grams in the generated text. We also utilize pretrained language model to measure fluency. Perplexity (PPL) is calculated using pretrained GPT-2 (Radford et al., 2019) without fine-tuning. The lower the perplexity is, the more fluent the generation is (based on GPT-2). "AvgLen" is the averaged word counts of the generated sequence. "failure" indicates the proportion of test sequences that fail to be generated at the first step (i.e., $\hat{Y}^0$ are all $[NOI]$'s). Finally, "AvgSteps" shows the average number of steps for the model to complete the generation. Note for GPT-2, the AvgSteps is based on tokens, while the AvgLen is based on words.

4.3 Experiment Results

Tables 2, 3, and 4 show the results of different models on the different datasets. On recall, GPT-2, due to its inability to enforce hard lexical constraints, yields the worst recall. For non-autoregressive models without ESAI, they still achieve high recall. Nevertheless, the high recall of POINTER-E is "contributed by" relatively high failure ratio ("failure") as recall is 1 even when the model fails to generate anything in the first stage. In other words, POINTER-E suffers from cold start problem. GREEDY ENCONTER and BBT-ENCONTER, in contrast, enjoy both good recall and zero failure ratio. With ESAI option, all non-autoregressive models can achieve perfect recall without much additional generation steps. However, this option does not reduce the high failure ratio of POINTER-E. On generation quality compared with human crafted text, GREEDY ENCONTER and BBT-ENCONTER outperform all other models by NIST, BLEU, and

[3] https://huggingface.co/

Method	Recall	NIST		BLEU		MTR	Entropy	DIST		PPL	AvgLen	failure	AvgSteps
		N-2	N-4	B-2	B-4		E-4	D-1	D-2				
Baselines													
GPT-2	0.70	1.38	1.39	0.13	0.08	0.19	4.91	0.16	0.57	**35.7**	201.4	**0.00**	256.84
Pointer-E	0.98	0.72	0.72	0.08	0.04	0.19	3.63	0.22	0.65	285.7	88.9	0.35	4.18
Pointer-E (+ESAI)	**1.00**	0.63	0.64	0.08	0.04	0.18	3.54	**0.23**	**0.67**	337.1	81.4	0.34	3.84
Enconter													
Greedy	0.96	1.95	1.96	0.19	0.09	**0.25**	**4.99**	0.16	0.58	112.1	192.5	**0.00**	17.56
Greedy (+ESAI)	**1.00**	**1.99**	**2.00**	**0.20**	**0.10**	**0.25**	4.95	0.16	0.59	111.7	181.9	**0.00**	16.48
BBT	0.94	1.83	1.84	0.19	**0.10**	0.23	4.87	0.18	0.62	154.9	161.1	**0.00**	8.19
BBT (+ESAI)	**1.00**	1.87	1.87	**0.20**	**0.10**	0.24	4.84	0.18	0.62	150.0	156.7	**0.00**	8.26
Human	-	-	-	-	-	-	4.99	0.20	0.66	53.0	202.1	-	-

Table 2: CoNLL-2003 result

Method	Recall	NIST		BLEU		MTR	Entropy	DIST		PPL	AvgLen	failure	AvgSteps
		N-2	N-4	B-2	B-4		E-4	D-1	D-2				
Baselines													
GPT-2	0.63	1.42	1.42	0.13	0.09	0.20	4.63	0.05	0.30	**66.3**	123.2	**0.00**	168.44
Pointer-E	0.99	1.69	1.70	0.20	0.12	0.28	3.84	**0.08**	**0.46**	901.2	73.4	0.38	5.01
Pointer-E (+ESAI)	**1.00**	1.71	1.72	0.21	0.12	0.28	3.85	**0.08**	**0.46**	966.4	74.1	0.34	4.91
Enconter													
Greedy	0.99	3.31	3.33	0.40	0.28	0.41	4.53	0.07	0.37	124.3	111.1	**0.00**	9.80
Greedy (+ESAI)	**1.00**	3.31	3.33	0.40	0.28	0.40	4.52	0.07	0.37	125.2	109.5	**0.00**	9.79
BBT	0.99	**3.59**	**3.62**	**0.45**	**0.34**	**0.44**	4.53	0.07	0.38	135.6	110.6	**0.00**	5.86
BBT (+ESAI)	**1.00**	3.55	3.57	0.44	**0.34**	**0.44**	**4.54**	0.07	0.38	137.7	111.5	**0.00**	5.92
Human	-	-	-	-	-	-	4.66	0.08	0.41	90.7	125.1	-	-

Table 3: SD result

Method	Recall	NIST		BLEU		MTR	Entropy	DIST		PPL	AvgLen	failure	AvgSteps
		N-2	N-4	B-2	B-4		E-4	D-1	D-2				
Baselines													
GPT-2	0.72	1.50	1.51	0.15	0.10	0.23	**4.40**	0.05	0.32	**101.0**	96.4	**0.00**	127.48
Pointer-E	0.98	1.32	1.32	0.17	0.10	0.26	3.46	**0.09**	**0.48**	2447.7	52.7	0.34	4.89
Pointer-E (+ESAI)	**1.00**	1.26	1.26	0.16	0.09	0.25	3.42	**0.09**	**0.48**	2535.7	53.3	0.38	5.07
Enconter													
Greedy	0.99	2.48	2.49	0.31	0.20	0.36	4.21	0.07	0.40	153.9	82.2	**0.00**	9.75
Greedy (+ESAI)	**1.00**	2.44	2.45	0.31	0.20	0.36	4.19	0.07	0.40	147.4	80.2	**0.00**	9.62
BBT	0.98	**2.73**	**2.74**	**0.34**	**0.24**	**0.38**	4.26	0.07	0.41	161.1	83.8	**0.00**	6.04
BBT (+ESAI)	**1.00**	2.69	2.70	**0.34**	0.23	**0.38**	4.25	0.07	0.41	157.5	83.6	**0.00**	6.05
Human	-	-	-	-	-	-	4.45	0.08	0.43	104.3	101.6	-	-

Table 4: SM result

MTR. This suggests that Enconter models learn the context of entities better compared to other models. On generation diversity, Pointer-E again has the highest diversity largely due to its high failure ratio. Finally, we discuss the efficiency of models measured by AvgSteps. The autoregressive nature of GPT-2 makes it the least efficient model among all. Pointer-E's ability to optimize masking patterns makes it the most efficient model. With balance binary tree reward, BBT-Enconter is able to finish its generation in fewer iterations than Greedy Enconter.

Case example Table 5 shows a case example from Jobs SM dataset. The entities of the given constraint are underlined. Invalid entities generated are colored in red, while the remaining ones are colored in blue. There are three types of invalid cases. First, the case of entity is not the same as specified. Second, the entity is not recalled in the generation. Third, the entity has its tokens separated by some other token(s). In this example, Pointer-E and Pointer-E ESAI terminate their generations prematurely. They fail to perform generation at the very first stage.

5 Related Work

Recent years have witnessed significant success using autoregressive (Dai and Le, 2015; Peters et al., 2018; Radford, 2018) generative models to conduct conditional generation on various tasks. CTRL (Keskar et al., 2019) uses control codes trained together with large amount of data to control the content to be generated. RecipeGPT (H. Lee et al., 2020) takes ingredients as a series of control and trains the generation of recipe text. PPLM (Dathathri et al., 2020)

GREEDY ENCONTER:
Degree / ACCA / CIMA / CA / CFA / F & B / Group Consolidation experience
Degree in General Accounting
Good track record in IFRSRSM, Fixed Assets, IFRS

GREEDY ENCONTER ESAI:
* * Degree / ACCA / CIMA / CA / CFA / CA / CAPA / Singapore Group Consolidation / Management / General Accounting experience
* Experience in IFRS or preferred
* Experience in Fixed Assets Management
* Extensive experience in IFRS

BBT-ENCONTER:
Degree or ACCA / CIMA or CFA qualifications
YEARSPAN'experience in handling Group s project / Consolidation / Good Inteconor / General Accounting
Knowledge of IFRSPAN, Fixed Assets (IFRS, etc)

BBT-ENCONTER ESAI:
Minimum Degree / ACCA / CIMA, CFA or equivalent
Minimum of YEARSPAN of experience in Marketing and Group Consolidation and General Accounting
Knowledge of IFRS) and Fixed Assets (IFRS)

GPT-2: (missing: IFRS, CFA, Group Consolidation, Fixed Assets)
Job Requirements :
- Degree in General Accounting / ACCA Qualification
- At least YEARSPAN of applicable working experience in similar capacity
- Must be able to multi-task and handle different priorities simultaneously
- General accounting knowledge will be advantageous
Interested applicant, kindly send in your CPA or CIMA reference number to EMAIL
EA Licence number : LICENSENUM
Registration number : REGNUM

Human:
Professional Qualifications : Bachelors Degree
Qualified with a professional financial body (ICAEW / ICPA / ACCA / CIMA / CFA etc)
Specialist Knowledge / Skills : Group Consolidation General Accounting IFRS Fixed Assets Industry Experience
Experience : YEARSPAN post qualified with extensive IFRS experience and industry experience

Table 5: A generated example from SM dataset. POINTER-E and POINTER-E ESAI are not shown since they failed to generate at first step.

directly steers the pretrained language by a bag-of-words model or simple linear discriminator. The above models in their own ways gain certain level of control over the content generation process. However, they do not provide a mechanism to directly enforce some lexical constraints in the final generation. Non-monotonic sequence generation (Welleck et al., 2019) is designed to perform hard lexical constrains generation based on binary tree structure. By leveraging level-order and in-order traversal of binary tree, the model allows text to be generated non-monotonically. Although the results from non-monotonic generation models seem promising, they do not perform token generation in parallel and the tree structure governing the generation process may produce many unused tokens during the generation.

The emergence of non-autoregressive language model provides another approach to support hard lexical constraints. Insertion transformer (Stern et al., 2019) uses transformer architecture with balanced binary tree loss to perform insertion-based generation. KERMIT (Chan et al., 2019) is proposed as a structure to unify insertion transformers. Levenshtein transformer (Gu et al., 2019) further introduces deletion as an action to take during generation. Our ENCONTER models differ from these previous models as they are not designed to support any lexical constraints, including entity constrains.

6 Conclusions

Constrained text generation is an important task for many real world applications. In this paper, we focus on hard entity constraints and the challenges associated with enforcing them in text generation. Our analysis of the state-of-the-art insertion transformers reveals issues, namely, cold start problems and inefficient generation. We therefore propose two insertion transformer models, GREEDY ENCONTER and BBT ENCONTER, that use a bottom-up preprocessing strategy to prepare training data so as to eliminate the cold start problem caused by top-down preprocessing strategy. BBT Enconter further incorporates a balanced tree reward scheme to make the generation process more efficient. Through experiments on real world datasets, we show that the two models outperform the strong baselines, POINTER-E and GPT2, in recall, quality and failure rate while not compromising much generation efficiency. For future research, it will be interesting to consider more diverse constraints (e.g., soft constraint, rules, etc.) and user interaction in the generation process to expand the scope of applications that can benefit from this research.

Acknowledgments

This research is supported by the National Research Foundation, Singapore under its International Research Centres in Singapore Funding Initiative. Any opinions, findings conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. 2020. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289.

William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. 2019. Kermit: Generative insertion-based modeling for sequences. *arXiv preprint arXiv:1906.01604*.

Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram cooccurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145.

Albert Gatt and Emiel Krahmer. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170.

David Gries. 1982. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2(3):207–214.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, pages 11181–11191.

Helena H. Lee, Ke Shu, Palakorn Achananuparp, Philips Kokoh Prasetyo, Yue Liu, Ee-Peng Lim, and Lav R Varshney. 2020. Recipegpt: Generative pretraining based cooking recipe generation and evaluation system. In *Companion Proceedings of the Web Conference 2020*, pages 181–184.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546.

J Edward Hu, Huda Khayrallah, Ryan Culkin, Patrick Xia, Tongfei Chen, Matt Post, and Benjamin Van Durme. 2019. Improved lexically constrained decoding for translation and monolingual rewriting. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 839–850.

Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL - A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint arXiv:1909.05858*.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the second workshop on statistical machine translation*, pages 228–231.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6834–6842.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324.

Lianhui Qin, Michel Galley, Chris Brockett, Xiaodong Liu, Xiang Gao, Bill Dolan, Yejin Choi, and Jianfeng Gao. 2019. Conversing by reading: Contentful neural conversation with on-demand machine reading. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5427–5436.

A. Radford. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985.

Jianheng Tang, Tiancheng Zhao, Chenyan Xiong, Xiaodan Liang, Eric Xing, and Zhiting Hu. 2019. Target-guided open-domain conversation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5624–5634.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Sean Welleck, Kianté Brantley, Hal Daumé, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *36th International Conference on Machine Learning, ICML 2019*, pages 11656–11676. International Machine Learning Society (IMLS).

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. 2018. Generating informative and diverse conversational responses via adversarial information maximization. In *Advances in Neural Information Processing Systems*, pages 1810–1820.

Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. 2020. Pointer: Constrained text generation via insertion-based generative pre-training. *arXiv preprint arXiv:2005.00558*.

Andrea Zugarini, Stefano Melacci, and Marco Maggini. 2019. Neural poetry: Learning to generate poems using syllables. In *International Conference on Artificial Neural Networks*, pages 313–325. Springer.