Ph.D. DISSERTATION

Effective Training Methods for Autoregressive Text Generation

자기회귀모델 기반 텍스트 생성을 위한 효과적인 학습 방법에 관한 연구

BY

KIM YANGHOON

AUGUST 2021

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Effective Training Methods for Autoregressive Text Generation

자기회귀모델 기반 텍스트 생성을 위한 효과적인 학습 방법에 관한 연구

지도교수 정 교 민

이 논문을 공학박사 학위논문으로 제출함

2021년 8월

서울대학교 대학원

전기 컴퓨터 공학부

김 양 훈

김양훈의 공학박사 학위 논문을 인준함

2021년 8월

위 원 장:	심 규 석
부위원장:	정 교 민
위 원:	이 상 구
위 원:	양 인 순
위 원:	임 성 수

Abstract

The rise of deep neural networks has promoted tremendous advances in natural language processing research. Natural language generation is a subfield of natural language processing, which is inevitable in building a human-like artificial intelligence since they take responsibility for delivering the decision-making of machines in natural language. For neural network-based text generation techniques, which have achieved most state-of-the-art performance, autoregressive methods are generally adapted because of their correspondence to the word-by-word nature of human language production. In this dissertation, we investigate two different ways to train autoregressive text generation models, which are based on deep neural networks. We first focus on a token-level training of question generation, which aims to generate a question related to a given input passage. The proposed Answer-Separated Seq2Seq effectively mitigates a problem from the previous question generation models that a significant proportion of the generated questions include words in the target answer. While autoregressive methods are primarily trained with maximum likelihood estimation, they suffer from several problems, such as exposure bias. As a remedy, we propose a sequence-level GAN-based approach for text generation that promotes collaborative training in both continuous and discrete representations of text. To aggregate the achievement of the research mentioned above, we finally propose a novel way of training a sequence-level question generation model, adopting a pre-trained language model, one of the most significant breakthroughs in natural language processing, along with Proximal Policy Optimization.

keywords: Deep Neural Networks, Text Generation, Question Generation

student number: 2014-22546

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Building a human-like artificial intelligence (AI) system has been a grand goal of human beings for an extended period of history. Great development of computer hardware such as graphics processing unit (GPU) results in the revival of AI systems, especially the rise of deep learning [8] based on artificial neural networks. Deep learning has changed classical AI systems a lot, from those rule-based systems supported by several hand-crafted features to end-to-end systems which are often represented as black-box models.

Natural language processing (NLP), one of the basic components of AI, is a medium that makes machines understand and generate human language. The advances in NLP research are gradually reflected in our real life; for example, Google Assistant is one of the most famous AI-powered virtual assistants to which almost every smartphone user can access. As shown in Figure 1.1, a series of complicated AI modules make up a whole assistant system, such as speech recognition [9, 10, 11, 12] for converting human voice to a written text, text classification [13, 14, 15, 16] to figure out what the user asks to do, and lastly, text generation [17, 4] to answer the user in human language rather than simply picking up from predefined sentences. As is described, text generation is an important subfield of NLP, which aims to generate textual output in natural language. Depending on the purpose of tasks, text generation can be categorized into

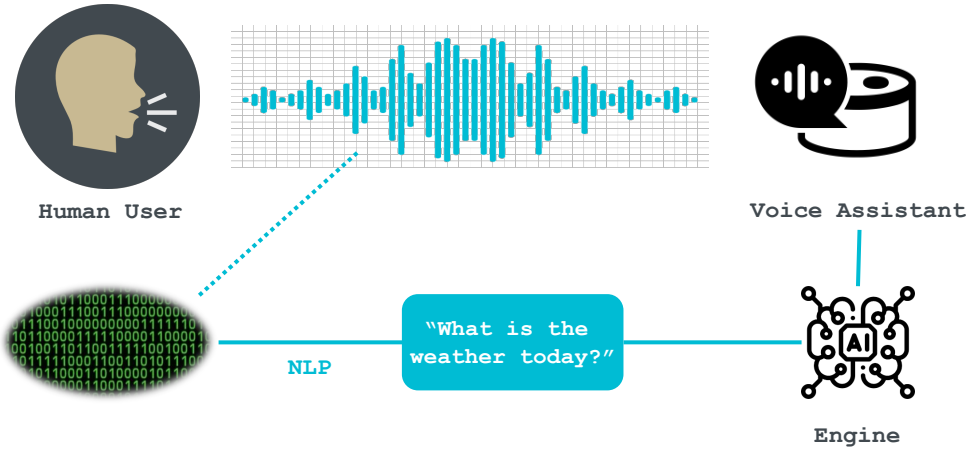

Figure 1.1: A virtual assistant comprises a series of AI modules, such as speech recognition, text classification, and text generation. With the help of deep learning, the interaction between each module is not limited to passing the processed features but is trained in an end-to-end fashion and works as a whole.

machine translation [18, 1, 19, 20, 2, 21], dialogue model [22, 23, 24, 25, 26, 27], text summarization [28, 29, 30, 31, 32] image captioning [33, 34, 35, 36] and etc. Basically, many of those text generation models take two steps in common: natural language understanding (NLU) and natural language generation (NLG).

In NLU, we first let a machine understand the meaning and structure of given text input. In general, text input written in natural language is first separated into tokens representing the (predefined) smallest units of text, such as words, characters, and morphemes. Subword-based tokenization is one of the frequently-used methods [37, 38, 39] these days since subwords help to find out the best trade-off between vocabulary size and the unit tokens. Followed by tokenization is the embedding of those tokens as vector representations. Embedding vectors are usually initialized with random normal distribution or certain distributional functions such as Xavier initialization [40]. The tokenization and the embedding process are shown in Figure 1.2. Finally, various type of neural network structures is adopted to extract contextual fea-

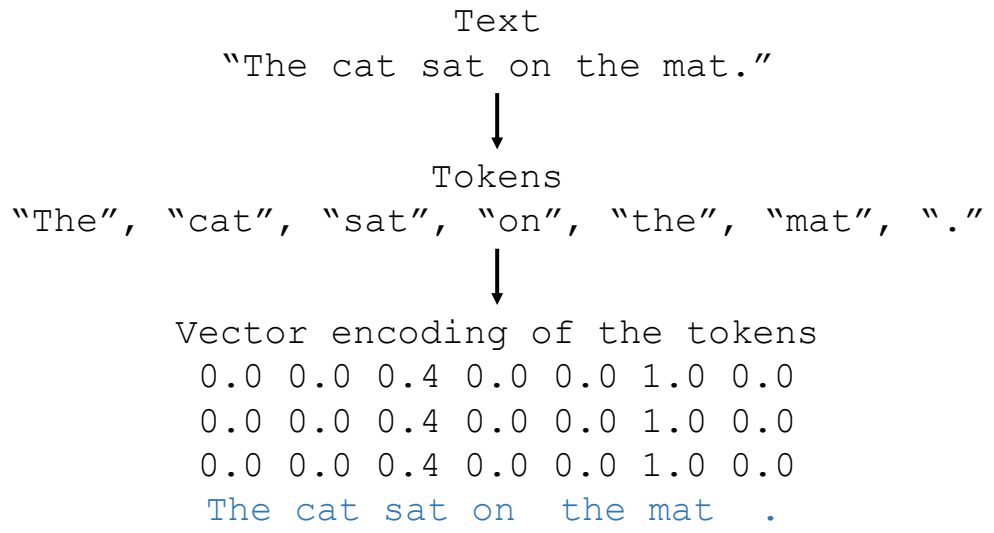

Figure 1.2: An example of word tokenization and vector embedding. The text input “The cat at on the mat.” is tokenized to a set of words and then embedded into 3-dimensional vector representations.

tures. Nowadays, pre-trained language models (PLM) [5, 41, 42, 43, 44, 45, 7] are mainly used as a feature extractor since they stand out against other models with the help of pre-trained knowledge.

After the dominant feature, which is related to the target task, is extracted with NLU, the corresponding response is generated during the NLG process. The network can either take an autoregressive fashion to generate a token per time-step or generate the whole response tokens at once in a non-autoregressive way [46, 47].

Various text generation models have been proposed under encoder-decoder [48, 18, 49] architectures which explicitly assign NLU and NLG to separate modules called encoder and decoder. Encoder and decoder are mostly based on neural networks that can model sequential data, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs). For a neural machine translation system as an example, as shown in Figure 1.3, the encoder takes charge of NLU, encapsulating the information from a human-written text as the internal state vectors. Then, the decoder interprets

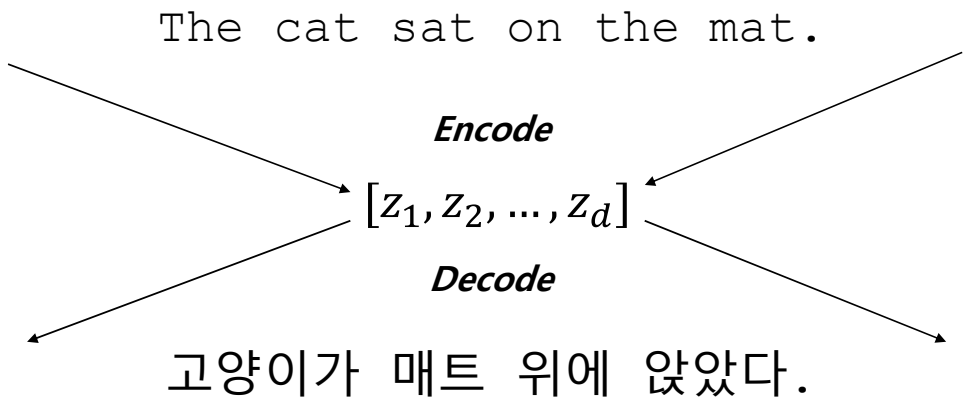

Figure 1.3: An example of an encoder-decoder structure of neural machine translation. The encoder converts the text input as vector representations. Then, the decoder deciphers the vectors to the target language.

the state vectors in the desired language, and this procedure represents NLG.

Textual data is treated as a type of sequential data since the human language is represented with a series of words, and the order of words matters in delivering the meaning. Since autoregressive approaches correspond to the word-by-word nature of human language production and can capture the target output distribution, they have prevailed in the early stage of modeling text generation. Figure 1.4 shows the mechanism of autoregressive models, which condition each output word on previously generated words. Beam search [50], an effective local search method for finding approximately optimal output, further strengthens autoregressive models and has achieved state-of-the-art performance on various generation tasks.

1.1 Contributions

This dissertation provides a comprehensive investigation into some efficient methods for training autoregressive text generation. From RNN-based encoder-decoder models to pre-trained language models, we explore various cutting-edge neural architectures

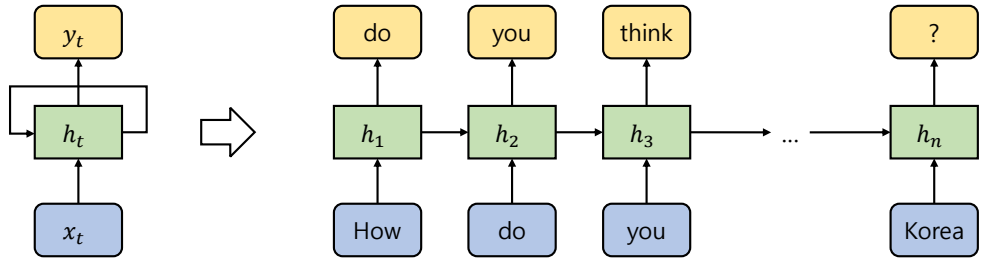

Figure 1.4: Autoregressive text generation models condition each output word on previously generated words.

for text generation. Besides the inspiration that artificial neural networks have similar architectural features to the biological neural networks, we are further inspired by human's way of learning text generation. We develop more human-like methods to train a neural text generation model, evaluating the generated sentence as a whole instead of evaluating each of generated tokens. Specifically, we focus on one practical text generation task, question generation. In the following lines, we explain each part of the contributions in detail.

First, we present our investigation of token-level training of an autoregressive question generation. Neural question generation (NQG) is the task of generating a question from a given passage with deep neural networks. Previous NQG models suffer from a problem that a significant proportion of the generated questions include words in the question target, resulting in the generation of unintended questions. In this study, we propose Answer-Separated Seq2Seq, which better utilizes the information from both the passage and the target answer. By replacing the target answer in the original passage with a special token, our model learns to identify which interrogative word should be used. We also propose a new module termed keyword-net, which helps the model better capture the key information in the target answer and generate an appropriate question. Experimental results demonstrate that our answer separation method significantly reduces the number of improper questions which include answers. Con-

sequently, our model significantly outperforms previous state-of-the-art NQG models.

Autoregressive text generation models are often trained with maximum likelihood estimation (MLE). Even though autoregressive methods trained with MLE have achieved great success, there still exist several drawbacks. The first problem is *exposure bias* [51]: during training, the model sequentially generates the next word depending on the ground-truth words; however, the model relies on its previously generated words at inference time. Therefore, the cumulative effect of incorrect predictions in the text sequence results in low-quality samples. The second problem lies in that the objective functions of MLE-based methods are rigorous [52]; the models are forced to learn every word in the target sentence. Under this strict guidance, the ability of language models to generate diverse samples can be severely limited.

In recent years, several approaches have been proposed to mitigate those shortcomings of sequence models trained with MLE methods. Among them, reinforcement learning (RL) [53] and generative adversarial networks (GANs) [54] especially have drawn attention as a remedy. Instead of getting feedback from the token-level loss during training, RL and GANs enable the sequence-level or phrase-level evaluation of the generated text.

In the initial period of GANs research, it has been considered that applying GANs to text-related tasks is not promising due to the discrete nature of the text. In the inference phase of the autoregressive text generation, the model iteratively samples the next word from the distribution of vocabulary. As this step includes the sampling process that hinders the backpropagation of the gradients from the discriminator, several approximation methods have been proposed to avoid the non-differentiability issue [55, 56, 17, 57]. GANs for text generation (text GANs) can be categorized into two groups, depending on the data space in which the GAN's discriminator operates: discrete-space methods and continuous-space methods. The discrete-space methods resolve the issue by employing RL and optimizing the next-word sampling policy directly in discrete action space. Such methods compute the rewards from complete sen-

tences and avoid error accumulation due to exposure bias. The continuous-space methods employ approximation techniques that map the text to continuous representation in order to circumvent the non-differentiable discrete process. Particularly, autoencoder-based methods effectively produce robust representations that can model complex discrete structures. To enhance the benefit from both methods, we propose a novel text GAN architecture that promotes the collaborative training of the continuous-space and discrete-space methods [4]. Our method employs an autoencoder to learn an implicit data manifold, providing a learning objective for adversarial training in a continuous space. Furthermore, the complete textual output is directly evaluated and updated via RL in a discrete space. The collaborative interplay between the two adversarial training effectively regularizes the text representations in different spaces. The experimental results on three standard benchmark datasets show that our model substantially outperforms state-of-the-art text GANs with respect to quality, diversity, and global consistency.

Finally, we explore the effect of sequence-level training methods on question generation. While pre-trained language models are the inevitable baselines for almost every NLP task these days, we first fine-tune T5 [7], one of the most potent encoder-decoder structured PLM, in an MLE manner. Then, we apply a sequence-level training method enhanced by Proximal Policy Optimization (PPO) [58, 59], an RL-based training algorithm, to fine-tune the model with task-specific reward further. The task-specific reward is given by another PLM-based question-answering (QA) model named SpanBERT [60].

The remainder of this dissertation is organized as follows. Chapter 2 provides several background knowledge related to text generation. In chapter 3, we explain the proposed Answer-Separated Seq2Seq model for question generation. Chapter 4 explains a GAN-based unconditional text generation model trained with sequence-level training. Further investigation of RL-based sequence-level training of the question generation model is presented in Chapter 5. Finally, the dissertation is concluded in Chapter 6.

Chapter 2

BACKGROUND

In this chapter, we briefly introduce the underlying knowledge on which the dissertation is based.

2.1 Sequence-to-Sequence model

Sequence-to-Sequence [37] (Seq2Seq) model is a family of machine learning approaches that are usually utilized as the baseline model for several NLP research. Seq2Seq model, as the name denotes, takes as input a word sequence and generates a new sequence that has some pre-defined relationship with the input sequence. In general, Seq2Seq models are composed of two modules - an encoder and a decoder. First, the encoder network is asked to understand the input sequence and make a vector representation (or a set of vectors) of it. Then, the decoder generates a desired sequence based on the encoded vector representation. Figure 2.1 shows an example of the RNN-based Seq2Seq dialogue model.

2.1.1 Sequence-to-Sequence model with Attention Mechanism

In a vanilla Seq2Seq model, the encoder tries to compress all of the information in the whole input sequence to a single vector. However, it is tough for a single vector to

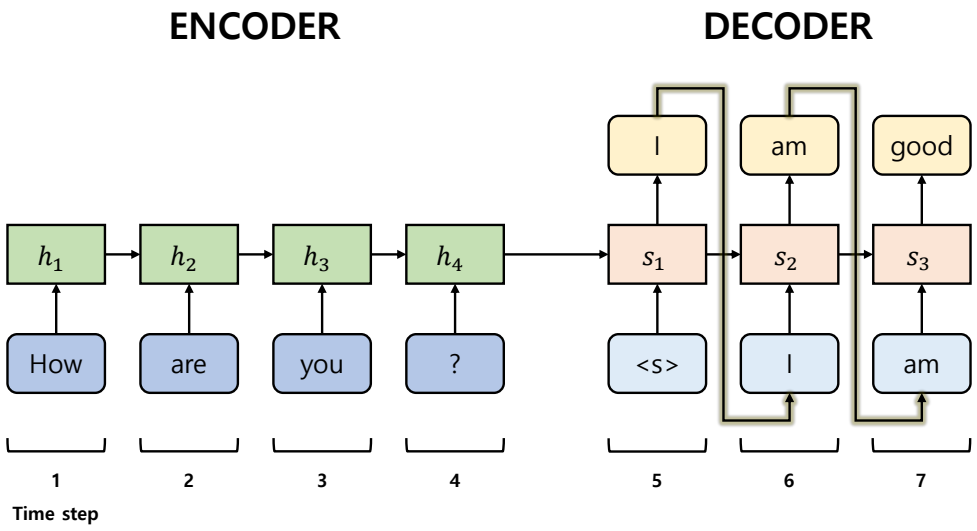

Figure 2.1: An example of the RNN-based sequence-to-sequence dialogue model.

represent the significant amount of important features in this procedure because of the complexity of the human language. Furthermore, since the single vector representation of the input sequence is usually the last hidden state of the encoder network, the early part of the encoded information is easily forgotten.

For the decoder, it is another challenge to decipher the extremely compressed vector representation of human language. In every decoding step, the decoder should generate a token relevant to the different parts of the input sequence. However, in the vanilla Seq2Seq setting, the decoder network can only refer to the last hidden state of the encoder, which means that the decoder should generate output sequence only based on "insufficient" and "over-compressed" vector representation.

Attention Mechanism [1, 61] is then proposed to address the fixed representation problem. As the name denotes, the attention mechanism aims to let the decoder network focuses on different parts of the input sequence at every decoding step. At each decoder step, the attention mechanism computes which part of the source sequence is more relevant to the current context of the output sequence. In this setting, the encoder does not have to compress the whole sentence input into a single vector - it gives vector

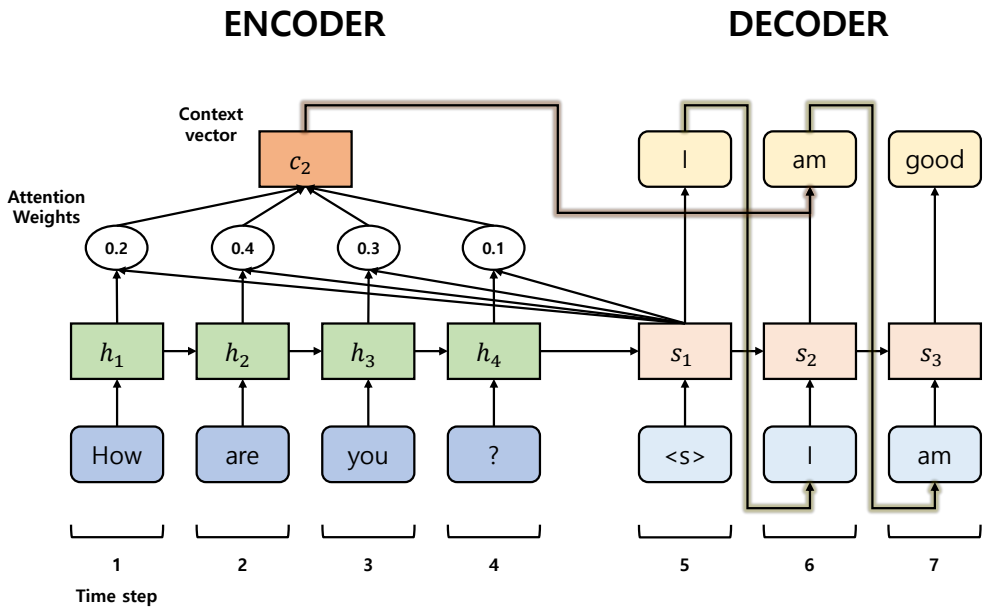

Figure 2.2: Graphical representation of how the Attention Mechanism [1] works.

representations for all input tokens. Attention Mechanism is formulated as:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.1}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{2.2}$$

$$e_{ij} = v^\top \tanh(W_a s_{i-1} + U_a h_j), \tag{2.3}$$

where h_j is j-th encoder hidden state, s_i is i-th decoder hidden state. c_i denotes the context vector, α_{ij} denotes the attention weights, and e_{ij} denotes an alignment score which represents how well the pair of input at position j and output at position i matches. Figure 2.2 shows the graphical representation of how the Attention Mechanism works.

2.2 Autoregressive text generation

Since the overall dissertation focuses on Seq2Seq-based text generation methods, we explain the autoregressive text generation in terms of Seq2Seq modeling. Given a text input $X = \{x_1, ..., x'_T\}$, an autoregressive Seq2Seq text generation model factors the distribution over possible output sentences $Y = \{y_1, ..., y_T\}$ into a chain of conditional probabilities with a left-to-right causal structure:

$$p_{AR}(Y|X;\theta) = \prod_{t=1}^{T} p(y_t|y_{0:t-1}, x_{1:T'};\theta), \tag{2.4}$$

Every conditional probability is parameterized as the single-step output of a neural network with a recurrence structure. Typically, an RNN-based encoder-decoder architecture [18] is used to capture the causal structure of the input/output distribution.

2.2.1 Maximum Likelihood Training

Factorizing the generated output distribution in an autoregressive manner enables straightforward maximum likelihood training with a cross-entropy loss function applied at each decoding step:

$$L_{ML} = -\log p_{AR}(Y|X;\theta) = \sum_{t=1}^{T} \log p(y_t|y_{0:t-1}, x_{1:T'};\theta). \tag{2.5}$$

Each conditional probability prediction is provided with direct supervision from the loss function.

2.2.2 Pros and cons of autoregressive methods

The autoregressive way of modeling sequence by conventional Seq2Seq text generation models has several benefits. Since the autoregressive factorization corresponds to the word-by-word nature of human language production, it can effectively capture the

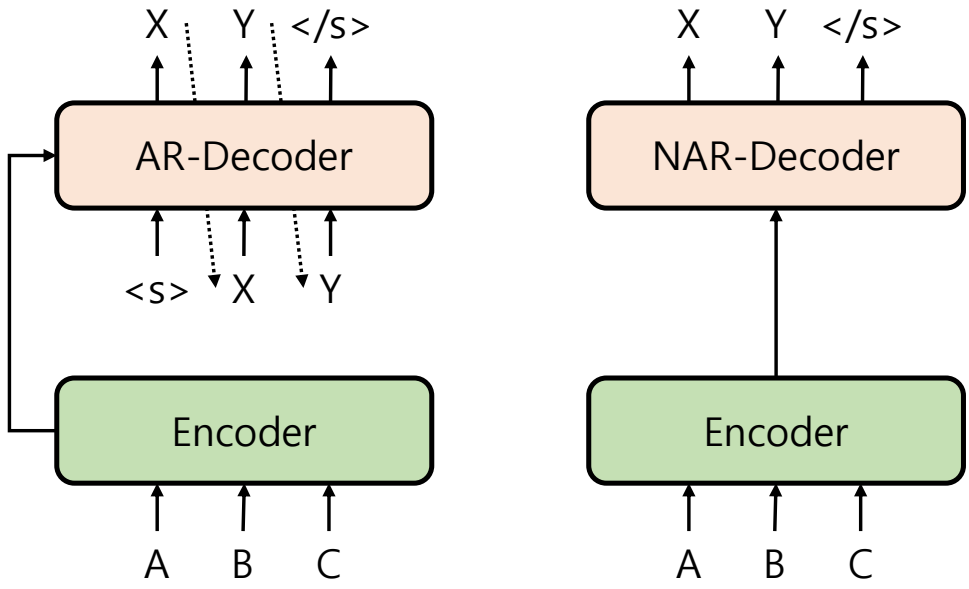

Figure 2.3: Comparison of the structural difference between autoregressive text generation and non-autoregressive text generation. Autoregressive model generates output tokens ("X", "Y", and "</s>") in a word-by-word manner, while the non-autoregressive model generates all output tokens in parallel.

distribution of output sequence. In addition, autoregressive models are easy to train. Beam search, an effective local search method for finding approximately optimal output sequence, further helps autoregressive models achieve state-of-the-art performance on most of the large-scale corpora of NLP research.

However, structural similarity to human language production does not always mean perfection. The word-by-word generation process of autoregressive decoding cannot be implemented in parallel, increasing training time. Meanwhile, beam search suffers from diminishing returns for increasing the beam size [62], and the introduction to the computational dependence between beams gives rise to limited search parallelism [46].

2.3 Non-autoregressive text generation

Non-autoregressive methods for text generation are then proposed to make up for those drawbacks of autoregressive methods. The most straightforward way of adopting non-autoregressive text generation is to remove the recurrent connection from an existing encoder-decoder model. Non-autoregressive modeling of text generation assumes that the target sequence distribution can be factorized into a product of separate conditional distribution p with p_L:

$$p_{NAR}(Y|X;\theta) = p_L(T|x_{1:T'};\theta) \cdot \prod_{t=1}^{T} p(y_t|x_{1:T'};\theta) \tag{2.6}$$

This modeling can still be trained with independent cross-entropy losses on each output distribution since it has an explicit likelihood function. What is more, these distributions can be computed in parallel at both training and inference time.

2.4 Transformers

[2] introduces a novel architecture called Transformer, a neural network architecture that completely relies on Attention Mechanism. Like Seq2Seq models, the original Transformer is a neural network-based encoder-decoder architecture for transforming an input sequence into another one. However, unlike most previous sequence models, Transformers do not rely on RNN-based architecture anymore.

RNNs used to be the most powerful structure to capture the sequential dependency until the emergence of Transformers family. Transformers then adopt a brand new way to base their main computation mechanism on the Attention Mechanism (described in Section 2.1.1), achieving state-of-the-art performance on various NLP tasks. A typical application of Transformer adoption in NLP is a language model called BERT [5].

The overall architecture of Transformer is illustrated in Figure 2.4. The main component of both encoder and decoder is a module that is mainly composed of a multi-

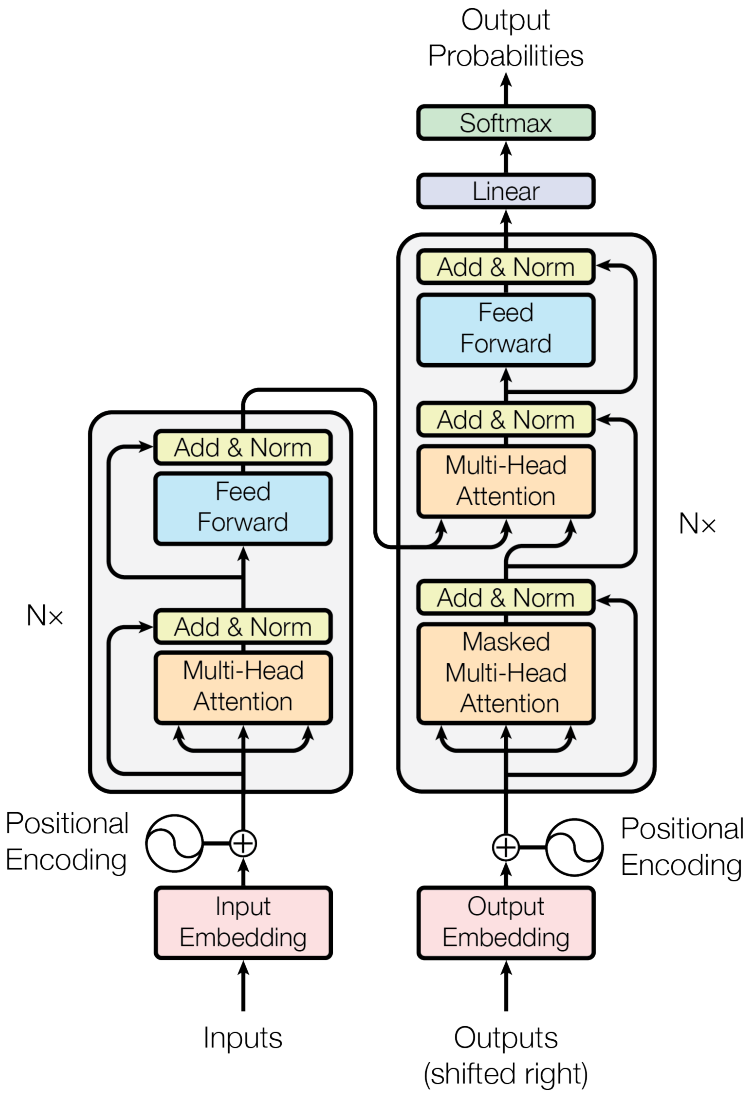

Figure 2.4: Overall architecture of Transformer [2].

head attention layer and a feed-forward layer. The module can be stacked on top of each other multiple times, enabling to form a deeper structure.

One slight but important part of the model is the positional encoding of the different tokens. Since the Attention Mechanism has no idea of the ordering of the input sequence, additional technique to remember the sequence order is required. Positional

encoding assigns every token in the sequence a feature vector representative of its relative position. These feature vectors are then added to the embedded representation of each word.

Attention mechanism in the Transformer (Scaled Dot-product Attention) can be seen as a variant of [61], and it is formulated as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.7}$$

where Q, K, and V are a query matrix, a key matrix, and a value matrix, respectively. d_k denotes the size of the feature dimension of the key matrix. The dot product between Q and K computes the similarity between each token in the encoder input sequence and decoder input sequence (encoder-decoder attention) or within a sequence (self-attention). After scaling and normalization using the softmax function, the similarity weights are then applied to all the tokens in the sequence introduced in V. Graphical illustration of the Scaled Dot-product Attention is shown in Figure 2.5 (left)

Instead of single attention functions with d_model-dimensional keys, values, and queries, Multi-Head Attention (Figure 2.5 (right)) is utilized, allowing the model to jointly attend to information from different representation subspaces at different positions:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.8}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \tag{2.9}$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. h is the number of parallel attention layers.

Transformer architecture has changed the entire NLP research paradigm from RNN-based / CNN-based methods to attention-based methods. One representative and the most significant change in NLP research since the propagation of Transformer is the adoption of pre-trained language models for various NLP tasks.

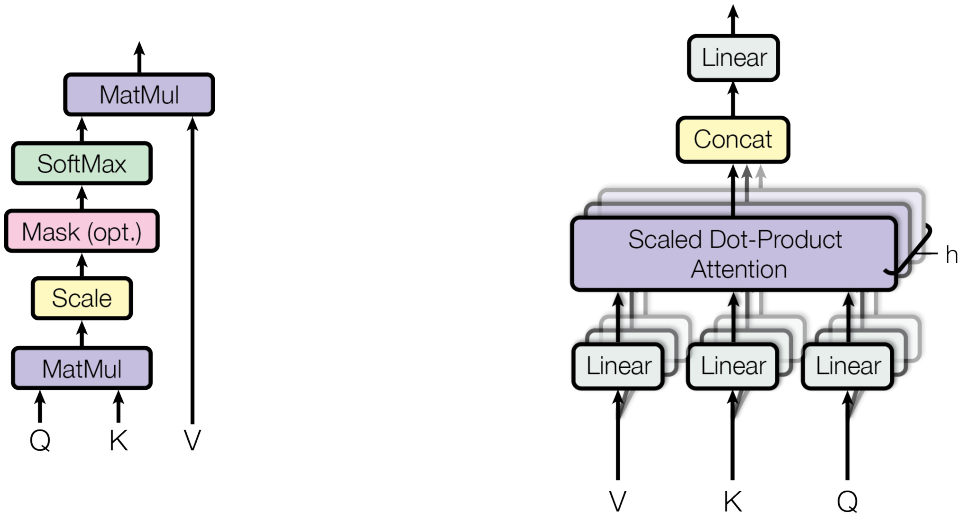

Figure 2.5: (left) Scaled Dot-product Attention. (right) Multi-head Attention consists of several attention layers running in parallel [2].

2.5 Reinforcement Learning

Reinforcement Learning (RL) is a kind of machine learning problem that aims at maximizing a long-term objective, the cumulative reward. The basic scenario of an RL system can be described as that an agent in state S_t which interacts with the environment via its actions A_t at discrete time steps and receives a reward R_t. Then, the agent transfers to a new state S_{t+1}. A canonical agent-environment feedback loop is depicted by Figure 2.6. The rationale behind RL is "trial and error", which is quite similar to how humans learn from the world.

Basically, many of RL theory is founded upon *the reward hypothesis* [53]: "That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (reward)". The reward here should be adequately defined depending on the final goal of a target task. For CartPole as an example, we can give a fixed-size positive reward for all the stable states and zero for losing balance.

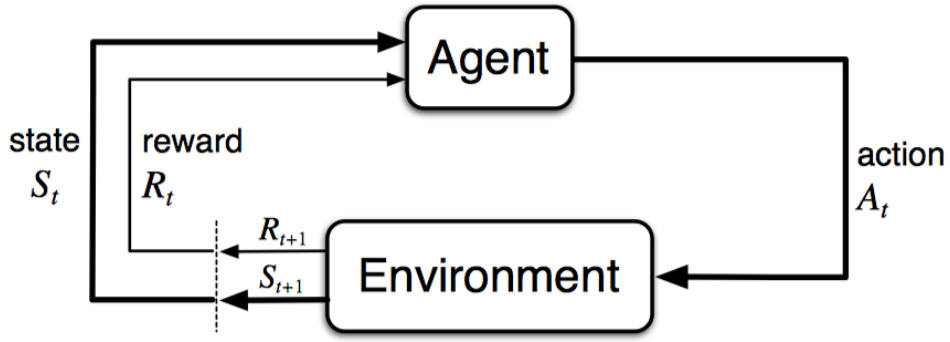

Figure 2.6: The canonical agent-environment feedback loop

Markov decision process (MDP) $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$can formally describe an environment for RL, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is a state transition probability matrix, $\mathcal{R}$ is a reward function, G_t is the total discounted reward from time-step t, and γ is a discount factor:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.10}$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2.11}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{2.12}$$

Thus, the entire interaction between the agent and the environment can be represented as a sequence of states, actions, and rewards known as a trajectory $<S_0,\ A_0,\ R_1,\ S_1,\ A_1,\ R_2,\ ... >$. MDP is based on the first-order Markov property: $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S1, S2, ..., S_t]$, which means that the future is independent of the past given the present.

2.5.1 Policy Gradient

Policy gradient methods are a type of reinforcement learning technique that explicitly models a parameterized function called policy π_θ and optimizes the expected return by gradient descent. A policy π_θ is defined as the probability distribution over actions

given states:

$$\pi_\theta(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{2.13}$$

The definition of the policy also indicates that the random dynamics of the environment that a specific action given a state does not mean forwarding to a particular state. For example, we can imagine standing on a windy field and fly a paper airplane. The plane is unlikely to fly in the direction we throw it.

The first step of the policy gradient is to reformulate the gradient of the objective function (one-step MDPs) with likelihood ratios:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] && (2.14)\\ &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a} && (2.15)\\ \nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a} && (2.16)\\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r] && (2.17) \end{aligned}$$

Policy gradient theorem enables the generalization of the likelihood approach to multi-step MDPs. We can directly change the instantaneous reward r with action-value function $Q^\pi(s,a)$:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(s,a) Q^\pi(s,a)], \tag{2.18}$$

where the expectation term can be approximated with Markov Chain Monte-Carlo (MCMC) or Temporal Difference (TD) learning.

Chapter 3

TOKEN-LEVEL TRAINING OF CONDITIONAL TEXT GENERATION MODEL

Neural question generation (NQG) is the task of generating questions from a given passage with deep neural networks. One of its key applications is to generate questions for educational materials [63]. It is also used as a way to improve question answering (QA) systems [64, 65, 66] or to engage chatbots to start and continue a conversation [67].

Automatic question generation (QG) from a passage is a challenging task due to the unstructured nature of textual data. One of major issues in NQG is how to take the question target, referred to as the target answer, in the passage. Specifying the question target is necessary for generating natural questions because there could be multiple target answers in the passage as in the following example. In Figure 3.1(a), the passage "John Francis O'Hara was elected president of Notre Dame in 1934." has various candidates to be asked such as the person "John Francis O'Hara", the location "Notre Dame", and the number "1934." Without taking the target answer as an additional input, existing NQG models such as [68] tend to generate questions without specific target. This is a fundamental limitation due to the fact that recent NQG systems mostly rely on RNN sequence-to-sequence model [18, 1], and RNNs do not have the

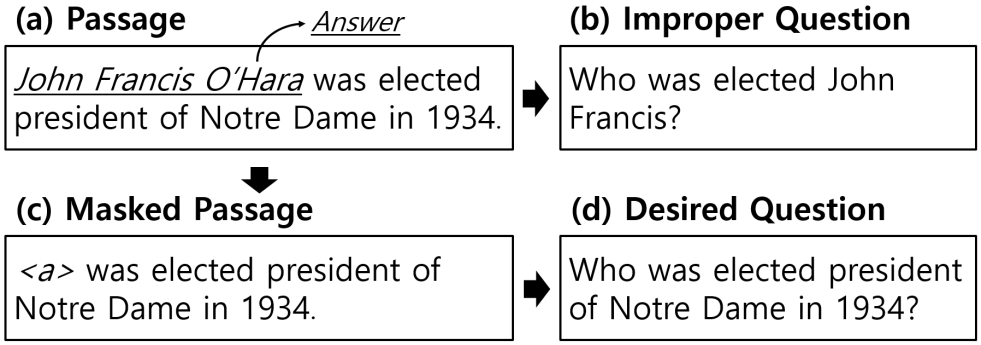

Figure 3.1: An example of overall idea for QG in this chapter. Generated questions from existing NQG models tend to include words from the answer, resulting in the generation of improper questions. Replacing the answer into a special token effectively prevents the answer words from appearing in the question, resulting in the generation of desired questions [3].

ability to model high-level variability [24].

To overcome this limitation, most recent NQG models incorporated the target answer information by using the answer position feature [69, 70]. However, these approaches have a critical issue that a significant proportion of the generated questions include words in the target answer. For example, Figure 3.1(b) shows the improperly generated question "Who was elected John Francis?"[1] which exposes some words in the answer. This problem results from the tendency of the sequence-to-sequence model to include all information from the passage [71]. It becomes severer with the recent trend that NQG models use the copy mechanism [72] to encourage that many words in the original passage appear in the question.

This study focuses on resolving this problem by separating the target answer from the original passage. For example, the masked passage "<a> was elected president of Notre Dame in 1934." in Figure 3.1(c) still contains enough information to generate the desired question in Figure 3.1(d), because the term "president" is mostly about some-

[1]This example is actually generated by our base model which will be introduced in the later part.

one's position. Interestingly, even though a target answer is replaced with a special token <a> in a passage, we can infer the interrogative word through contextual information from the remaining part of the passage. Therefore we expect that separating a target answer will prevent the answer inclusion problem.

In this chapter, we develop a novel architecture named **Answer-Separated Seq2Seq** which treats the passage and the target answer separately for better utilization of the information from both sides. The first step in the proposed NQG model is an answer masking. Literally, we replace the target answer with the mask token <a>, and keep the corresponding target answer apart. The masked passage is encoded by an RNN encoder inside of our model. This approach to separate the target answer from the passage helps our model to identify the question type related to the target answer because the model learns to capture the position and contextual information of the target answer with the help of the token <a>. Furthermore, we propose a new module called **keyword-net** as a part of Answer-Separated Seq2Seq, which extracts key information from the target answer kept apart before. The keyword-net makes our NQG model be consistently aware of the target answer, supplementing the information deficiency caused by answer separation. This module is inspired by how people keep the target answer in mind when they ask questions. Lastly, we adopt a retrieval-style word generator proposed by [73] which better captures the word semantics during the generation process.

When we evaluate our Answer-Separated Seq2Seq on the SQuAD dataset [74]), our model outperforms previous state-or-the-art NQG models by a considerable margin. We empirically demonstrate the impact of the answer separation in following three ways: the rare appearance of the target answer in the generated questions, the better prediction of interrogative words, and the higher attention weights of the <a> token to interrogative words. Furthermore, trained with the only questions generated by our model, a machine comprehension system achieves a comparable results.

3.1 Related Work

Recently, there have been several NQG models which are end-to-end trainable from (*passage*, *question*, *answer*) triplets written in natural language. [68] first dealt with end-to-end learning with regard to the question generation problem using a sequence-to-sequence model with an attention mechanism, achieving better performance than rule-based question generation methods in both automatic and human evaluations. However, their model did not take the target answer into account, resulting in generation of the questions which were full of randomness.

To generate more plausible questions, [69] utilized answer positions to make the model aware of the target answer and used NER tags and POS tags as additional features. [70] utilized the multi-perspective context matching algorithm of [75] to employ the interaction between the target answer and the passage for collecting the relevant contextual information. Both works employed a copy mechanism [72] to reflect the phenomenon by which many of the words in the original passage are copied to the generated question. However, none of them dealt with the issue of many of generated questions including target answers, and the copy mechanism could intensify this problem. To tackle this problem, this chapter focuses on developing an NQG model that utilizes the target answer as a separated knowledge.

Additionally, there have been several works which utilize question generation to improve the question answering system. [64] crawled an external QA dataset and generated questions from it through their retrieval-based and generation-based question generation methods. With the generated questions as additional data for training the QA system, they demonstrated that their question generation model helps to improve QA systems. More recently,[66] presented a joint training algorithm that improves both the question answering system and the question generation model.

To the best of our knowledge, none of the previous works has focused on the issue that a significant proportion of generated questions include words in the target answers.

3.2 Task Definition

Given a passage $X^p = (x_1^p, ..., x_n^p)$ and a target answer $X^a = (x_1^a, ..., x_m^a)$ as input, the NQG model aims to generate a question $Y = (y_1, ..., y_T)$ asking about the target answer X^a in the passage X^p. The NQG task is defined as finding the best $\overline{Y}$ that maximizes the conditional likelihood given the X^p and the X^a:

$$\overline{Y} = \underset{Y}{\operatorname{argmax}} P(Y|X^p, X^a) \tag{3.1}$$

$$= \underset{Y}{\operatorname{argmax}} \sum_{t=1}^{T} P(y_t|X^p, X^a, y_{<t}) \tag{3.2}$$

3.3 Base Model: Encoder-Decoder with Attention

Following previous works, we base our model on the RNN encoder-decoder architecture [18], which is an RNN-based sequence-to-sequence learning model. It generates a task-specific sequential output from a given sequential input and is widely adopted in sequence generation tasks such as neural machine translation [18, 1], text summarization [29] and dialogue model [22, 24]. In neural question generation, the model takes a passage X^p as an input and outputs a question Y which is relevant to the input passage X^p. Note that the base model does not take the target answer as the input.

An RNN encoder-decoder model consists of two parts: an encoder and a decoder. The encoder is used to represent the variable-length input sequence as a fixed-length vector which includes contextual features of the input sequence, reflecting dependency among each input token. The decoder then generates an output sequence based on the encoder output.

In general, attention mechanism [1], which functions as visual attention mechanisms found in human, is combined with the encoder-decoder model. The mechanism alleviates the bottleneck that the decoder only relies on a fixed-size vector to generate sequences. With the attention mechanism, the decoder is able to pay attention to the

most relevant parts of the given input sequence while generating an output sequence. In the following section, we describe the structure of the encoder-decoder with attention in details.

Encoder

The encoder is used to extract contextual features from the given input passage X^p. We use an one-layer bi-directional LSTM as the encoder. A bi-directional LSTM consists of a forward LSTM and a backward LSTM:

$$\overrightarrow{h_i^p} = \overrightarrow{LSTM}(x_i^p, \overrightarrow{h^p}_{i-1}) \tag{3.3}$$

$$\overleftarrow{h_i^p} = \overleftarrow{LSTM}(x_i^p, \overleftarrow{h^p}_{i+1}) \tag{3.4}$$

$$h_i^p = [\overrightarrow{h_i^p}; \overleftarrow{h_i^p}] \tag{3.5}$$

For each time step i, forward hidden state $\overrightarrow{h_i^p}$ and backward hidden state $\overleftarrow{h_i^p}$ are concatenated to form a hidden state of bi-LSTM.

Decoder

Given the extracted features from the encoder, the decoder generates the corresponding question Y. We utilize a one-layer uni-directional LSTM with attention:

$$s_t = LSTM(y_{t-1}, s_{t-1}, c_t) \tag{3.6}$$

$$P(y_t|y_{<t}, X^p) = Softmax(W_o s_t) \tag{3.7}$$

For each time step t, the output token of previous time step y_{t-1}, the hidden state of previous time step s_{t-1} and the context vector of current time c_t are passed through the decoder LSTM to compute the decoder hidden state of current time step s_t. Each hidden state of decoder s_t is then linearly projected with a trainable weight matrix W_o and passed through a softmax layer to compute the probability of output y_t. The context vector c_t is used to reflect the most relevant feature from the input passage X^p

while generating the current question token y_t. In Eq. (3.8), the alignment score e_{ti} is computed as the matching score between s_{t-1} and h_i^p, where W_c and U_c are trainable weight matrices and $v^\top$ is a trainable vector. As in Eq. (3.9), the alignment weight α_{ti} is computed with normalization and we take the weighted average of h_i^p as context vector:

$$e_{ti} = v^\top \tanh(W_c s_{t-1} + U_c h_i^p) \tag{3.8}$$

$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{n} exp(e_{tk})} \tag{3.9}$$

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i^p \tag{3.10}$$

3.4 Answer-Separated Seq2Seq

Previous encoder-decoder based neural question generation models take the whole passage X^p as an input. However, RNN encoders tend to pass all of the information in the passage to the decoder, causing a serious issue: the generated question often includes the target answer X^a. Therefore, we propose **Answer-Separated Seq2Seq** which treats the target answer and the passage separately for better utilization of the information from both sides. With a simple pre-processing of data, we separate the target answer from the input passage. Encoded with two individual encoders of Answer-Separated Seq2Seq, contextual feature of the passage and the target answer are passed to the decoder. We further propose **keyword-net** as another part of Answer-Separated Seq2Seq, which is used to extract the key information from target answer. In every decoding step, the decoder utilizes both the contextual feature of the passage from the attention mechanism and the keyword feature of the target answer from the keyword-net to generate a question that is related to the target answer in the passage. Furthermore, we adopt a retrieval style word generator by [73] as the output layer of the decoder to better capture the word semantics.

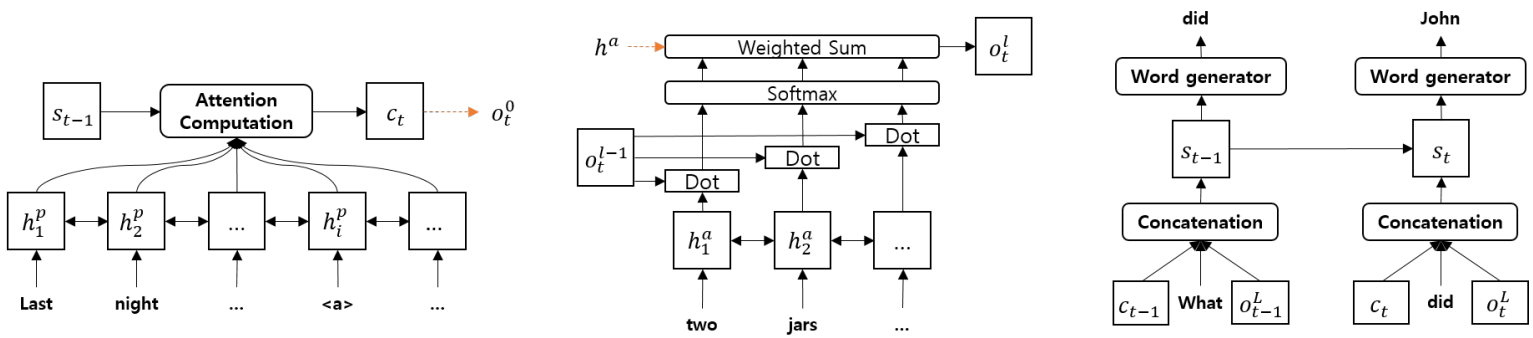

Figure 3.2: Overall architecture of the proposed model [3].

Different from the general RNN encoder-decoder, our Answer-Separated Seq2Seq consists of :

- Two encoders each to extract contextual feature from the passage X^p and the target answer X^a.
- Answer-Separated decoder which combines both the information from the passage and the target answer.

In the following section, we give a detailed description on how Answer-Separated Seq2Seq works. An overview of our Answer-Separated Seq2Seq is shown in Figure 3.2.

3.4.1 Encoder

Answer-Separated Seq2Seq contains two individual encoders each for encoding the passage X^p and the target answer X^a. Similar to the base model, we use two one-layer bi-LSTMs as encoders.

Answer-Separated Passage Encoder

Rather than feeding the passage encoder with additional features to emphasize the answer position, we first extract the target answer inside the passage and simply replace the corresponding target answer with a special <a> token as in Figure 3.1(c). In this way, the model learns to capture the position and contextual information of the target answer, knowing which part of the passage should be focused by the generated question. As a result, the probability that the generated question includes the target answer is reduced. This has a direct effect of preventing generation of questions irrelevant to the given target answer. We use the same one-layer bi-LSTMs as in Eq. (3.1) and Eq. (3.2).

Answer encoder

We use another one-layer bi-LSTM to encode the target answer X^a. In the last time step of the answer encoder, the hidden state of each LSTM is concatenated to form the final hidden state h^a_{final} , which represents the overall feature of the answer X^a:

$$\overrightarrow{h^a_j} = \overrightarrow{LSTM}(x^a_j, \overrightarrow{h^a}_{j-1}) \tag{3.11}$$

$$\overleftarrow{h^a_j} = \overleftarrow{LSTM}(x^a_j, \overleftarrow{h^a}_{j+1}) \tag{3.12}$$

$$s_0 = h^a_{final} = [\overrightarrow{h^a_m}; \overleftarrow{h^a_m}] \tag{3.13}$$

3.4.2 Answer-Separated Decoder

To exploit sufficient information from both the passage and the target answer, we design the Answer-Separated decoder. Based on LSTM, Answer-Separated decoder employs features of the passage and the target answer in the following three ways.

Decoder Initialization

We initialize the decoder state with the final answer vector h^a_{final}.

Incorporating the Key Feature of the Answer

We extract the key information in the target answer to disambiguate the question target. For example, given a passage "Steve Jobs is the founder of Apple" and the target answer "founder of Apple", we want to generate a question like "Who is Steve Jobs?". Then "founder" in "founder of Apple" is a keyword which defines the representative characteristic of the whole answer. In every decoding step, we use an attention-based module, termed **keyword-net**, to extract the key information from the target answer. For each layer of the keyword-net, a normalized matching score between output vector of last layer o^{l-1}_t and answer hidden states h^a_j is computed. We then take the weighted average over h^a_j as the extracted keyword feature o^l_t in current layer l. We initialize

o_t^0 with context vector c_t of current decoding step. Following equations describe the mechanism of keyword-net:

$$o_t^0 = c_t \tag{3.14}$$

$$p_{tj}^l = Softmax((o_t^{l-1})^\top h_j^a) \tag{3.15}$$

$$o_t^l = \sum_j p_{tj}^l h_j^a \tag{3.16}$$

$$s_t = LSTM(y_{t-1}, s_{t-1}, c_t, o_t^L) \tag{3.17}$$

Retrieval Style Word Generator

Based on the current decoder structure, we further adopt an architecture which can generate words by querying distributed word representation, with the purpose of capturing the semantic information of the according words.

[73] proposed a retrieval style word generation layer which can remedy a shortcoming of the sequence-to-sequence model that sequence-to-sequence model has tendency to memorize the sequence pattern rather than reflecting word meanings. They made use of word embeddings to tackle the problem. Their word generator produces words by querying the distributed word representations, hoping to capture the meaning of used words. We then borrow the idea behind this novel word generator to replace the existing output layer in our decoder.

The query q_t is computed as a combination of the decoder hidden state s_t and the context vector c_t. By querying q_t to each of the word embedding e_k, we can compute the relevance score between q_t and e_k where W_a is a trainable parameter matrix. Then the normalized value of score function denotes the generation probability of each word. Since the original output layer takes the most of model parameters, we can dramatically reduce the parameter size and the time of model convergence by using this

word retrieval layer:

$$q_t = \tanh(W_q[s_t; c_t]) \tag{3.18}$$

$$score(q_t, e_k) = q_i^\top W_a e_k \tag{3.19}$$

$$p(y_t) = Softmax(score(q_t, e_k)) \tag{3.20}$$

3.5 Experimental Settings

In this section, we first introduce the dataset we conduct experiments on. Then we give a detailed description of hyperparameter settings of our model. Lastly, several evaluation methods mainly used to assess the quality of generated questions are introduced.

3.5.1 Dataset

For fair comparison, we use the same dataset that is used by previous works [68, 69, 70]: two processed versions of SQuAD[74] dataset. The original SQuAD dataset contains 23,215 paragraphs from 536 articles with over 100k questions and their answers, which are originally created by crowd-workers. Since the original dataset is divided into train/dev splits, [68, 69] re-divided them into train/dev/test splits, and extracted passages from the paragraph that contains the target answer, each of which we call data split-1 and data split-2 in the following lines. For the data split-1, since [68] does not include the target answers, [70] extracted them from each passage to make (*passage*, *question*, *answer*) triplets. As a result, data split-1 and data split-2 contains 70,484/10,570/11,877 triplets and 86,635/8,965/8,964 triplets respectively. We tokenize both data splits with Stanford CoreNLP [76] and then lower-case them.

3.5.2 Implementation Details

We implement our models in Tensorflow 1.4 and train the model with a single GTX 1080 Ti. The hyperparameters of our proposed model are described as follows.

Our model consists of two one-layer encoders each for encoding passages and target answers, and a one-layer decoder to generate questions. The number of hidden units in both encoders and the decoder are 350. For both encoder and decoder, we use 34k most frequent words appeared in training corpus, replacing the rest with the <UNK> token. We use 300-dimensional pre-trained GloVe [77] embeddings trained on 6 billion-token corpus for initialization and freeze it when training. Weight normalization is applied to the attention module and dropout with P_{drop} = 0.4 is applied for both RNNs and the attention module. The layer size of keyword-net is set as 4.

Training and Inference

During training, we optimize the cross-entropy loss function with the gradient descent algorithm using Adam [78] optimizer, with an initial learning rate of 0.001. The mini-batch size for each update is set as 128 and the model is trained for up to 17 epochs.

When testing, we conduct beam search with beam width 10 and length penalty weight 2.1. Decoding stops when the generated token is <EOS>. The Performances of all our models are reported as mean and standard derivation values (Mean $\pm$ Std).

Named Entity Replacement

To further improve the model performance, we pre-process the data with a very simple technique. Since most named entities do not appear often, by replacing those named entities with representative tokens, we can not only reduce unknown words but also capture the grammatical structure. We look for the named entity tags for tokens in the given passage and replace each of them with the corresponding tag. We make sure that the same entity is assigned the same tag. NER tags are extracted with named entity tagger in Stanford CoreNLP. For those passages that have different named entities with the same tag, we distinguish them with different subscripts such as $Person_1, Person_2$. We store a matching table between named entities and tags, which is used to post-process the generated questions.

3.5.3 Evaluation Methods

Following [69, 70], we compare the performance of NQG models with 3 evaluation metrics: BLEU-4, Meteor and Rouge$_L$, which are standard evaluation metrics of machine translation and text summarization. We use the evaluation package published by [79].

BLEU-4

BLEU-4 measures the quality of the candidate by counting the matching 4-grams in the candidate to the 4-grams in the reference text.

Meteor

Meteor compares the candidate with the reference in terms of exact, stem, synonym, and paraphrase matches between words and phrases.

Rouge$_L$

Rouge$_L$ assesses the candidate based on longest common subsequence shared by both the candidate and the reference text.

3.6 Results

3.6.1 Performance Comparison

We compare our model with previous state-of-the-art NQG models. Since there exists two different data splits processed by [68, 69], we conduct experiments on both data splits. To figure out the effect of each module, we also conduct ablation tests against some key modules: **ASs2s** denotes the complete Answer-Separated Seq2Seq model. **ASs2s-$<$ a$>$** is the Answer-Separated Seq2Seq without replacing the target answer in the original passage. **ASs2s-keyword** is the Answer-Separated Seq2Seq without

keyword-net. **ASs2s-ASdec** is the Answer-Separated Seq2Seq without the Answer-Separated decoder but with a general LSTM decoder.

Table 3.1: Evaluation of the proposed model and previous NQG models with three metrics: BLEU-4, METEOR and ROUGE-L

Model	Split-1			Split-2
	BLEU	METEOR	ROUGE-L	BLEU
NQG [68]	12.28	16.62	39.75	-
MPQG [70]	13.98	18.77	42.72	13.91
NQG++ [69]	-	-	-	13.29
ASs2s-ASdec	12.30 ± 0.26	16.70 ± 0.22	40.32 ± 0.26	12.25 ± 0.24
ASs2s-keyword	13.95 ± 0.29	19.34 ± 0.24	40.60 ± 0.25	13.86 ± 0.30
ASs2s-<a>	14.37 ± 0.28	18.95 ± 0.24	42.06 ± 0.27	14.05 ± 0.30
ASs2s	$\mathbf{16.20 \pm 0.32}$	$\mathbf{19.92 \pm 0.20}$	$\mathbf{43.96 \pm 0.25}$	$\mathbf{16.17 \pm 0.35}$

As shown in Table 3.1, ASs2s outperforms all of the previous NQG models on both data splits by a great margin, showing that separate utilization of target answer information plays an important role in generating the intended questions. With the help of Answer-Separated decoder, ASs2s-<a> still outperforms the previous NQG models except for ROUGE-L on data split-1. However, there is a considerable decrease in all metrics compared to the complete model. This results from the fact that answer separation prevents generated question from including the answer. Similarly, ASs2s-keyword has a big drop in performance and this verifies that the keyword-net has actual impact on improving the performance. ASs2s-ASdec has greater decrease in all metrics compared to the ASs2s. This is a very natural result because without the Answer-Separated decoder, the model has to generate questions by only relying on context around the target answer position without knowledge of the target answer.

Table 3.2: Percentage of complete/partial inclusion of the target answer in generated questions

Model	Complete	Partial
seq2seq+AP	0.8%	17.3%
MPQG [70]	2.9%	24.0%
ASs2s	**0.6%**	**9.5%**

3.6.2 Impact of Answer Separation

Answer separation helps the model generate the right question for the given target answer. Since the base model does not utilize the target answer information, we further define **Seq2Seq+AP(Answer Position)** as base model with answer position feature [69] for comparison. We show the benefits of Answer-Separated Seq2Seq in three aspects.

Answer Copying Frequency

If a NQG model captures the question target well, the generated question will rarely include the target answer. We verify the assumption by computing the percentage of generated questions including target answers. Since [68] ignores the target answer, we choose Seq2Seq+AP to represent [68] with answer position feature. Further, we choose the previous state-of-the-art [70] for comparison because both [69] and [70] use the copy mechanism.

As shown in Table 3.2, the percentage that the target answers are either completely or partially included in the generated questions is significantly lower in our model. We also figure out an interesting observation: even though [70] is the previous state-of-the-art NQG model, it generates more irrelevant questions to the target answer when compared to Seq2Seq+AP. This observation indicates the negative effect of copy mechanism that the target answer inside the passage is unintentionally copied to the

generated question.

Interrogative Word Prediction

To figure out the effect of Answer-Separated Seq2Seq on question type prediction, we compare the recall of each interrogative word prediction between the generated questions of Answer-Separated Seq2Seq and Seq2Seq+AP. We group questions into 8 categories: "what", "how", "when", "which", "where", "who", "why" and "yes/no". As shown in Table 3.3, Answer-Separated Seq2Seq has better recall score over Seq2Seq+AP in all categories. Especially, the recall of question types "how", "when", "where" and "who" improved in big magnitude. Both model's recall of question type "what" is very high because "what" takes up more than half of the whole training set (55.4%). Both model's recall of type "which" is very low. This may result from the fact that some combinations like "which year" and "which person" may be generated as "where" and "who" respectively. For question types "why" and "yes/no" which only take up 1.5% and 1.2% of the training set respectively, both models did not perform well due to the small amount of data.

Table 3.3: Recall of interrogative word prediction

Model	Question type							
	what	how	when	which	where	who	why	yes/no
seq2seq+AP	77.3%	56.2%	19.4%	3.4%	12.1%	36.7%	23.7%	5.3%
ASs2s	82.0%	74.1%	43.3%	6.1%	46.3%	67.8%	28.5%	6.2%

Attention from <a>

We verify the effect of replacing answer with <a> by comparing attention matrices. Given the passage "john francis o'hara was elected president of notre dame in 1934." and the target answer "john francis o'hara", following Figure 3.3(a) and Figure

3.3(c) show the attention matrices produced by our Answer-Separated Seq2Seq and Seq2Seq+AP respectively.

As shown in Figure 3.3(a), the interrogative word ”who” gets most of the attention weights(higher attention weights) from the <a> token in our Answer-Separated Seq2Seq. Further more, Our model can generate a question that is exactly related to the target answer. With additional answer position features as in Figure 3.3(c), only a part of answer is attended while generating the interrogative word “who”. In this case, if the answer has some contextual information, then the model may omit it, generating an unintended question. Also, the generated question contains “john francis” which is a part of the target answer. We infer that the encoder tends to utilize more information from the word embeddings rather than answer position features, since the word embedding has far more information than answer position features.

3.6.3 Question Generation for Machine Comprehension

By training a machine comprehension system on the synthetic data generated by our model, we verify that our model has an enough ability to generate natural and fluent questions. By changing the position of the <a> token, we can easily produce various questions with our model. Figure 3.3(a) and Figure 3.3(b) shows one example where we use our model to generate two different questions corresponding to different target answers from the same input passage.

We experiment with QANet [80] on SQuAD dataset to verify whether the generated questions from our model are valid or not. Since most of the answers correspond to named entities, we use words and phrases that are named entities from training part of data split-1 as target answers. Then, we pair those answers with corresponding passages. We also make sure that selected answers are not overlapped with answers in the original SQuAD dataset because our NQG model is trained with the target answer provided with SQuAD dataset. If answers are overlapped, our model may generates exact the same questions as the golden questions. then we pair those answers with

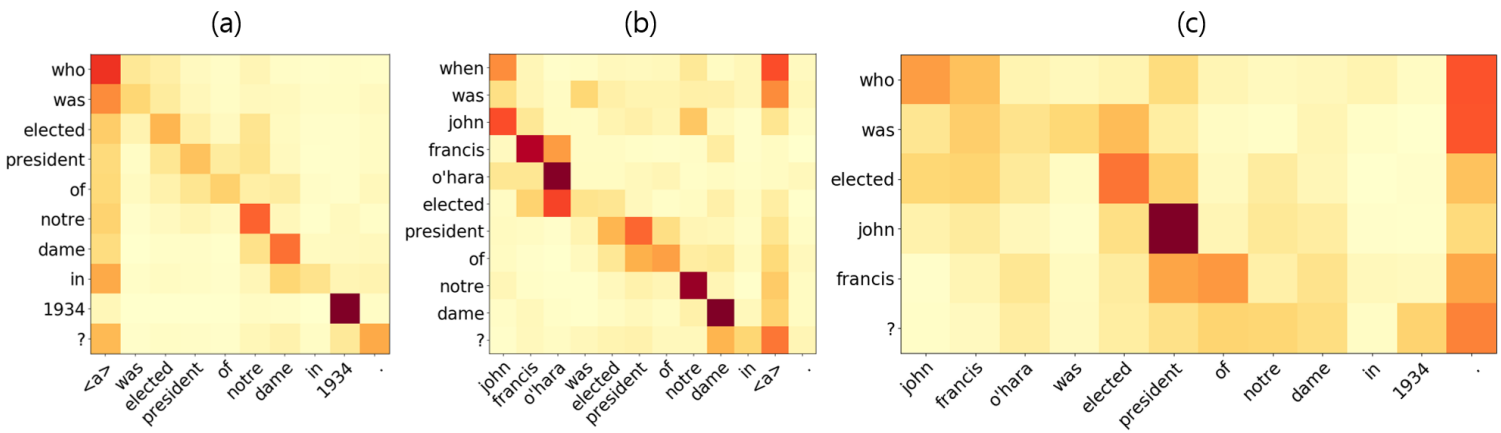

Figure 3.3: (a) and (b) show attention matrices of the proposed model given a passage with two different target answers. (c) shows an attention matrix of Seq2Seq+AP given the same passage and the target answer as (a) [3].

Table 3.4: Performance of the machine comprehension system which is trained only with synthetic data generated by the proposed model

Answers	Exact Match (EM)	F1 score
ALL	22.72	31.58
NER	49.09	56.57

corresponding passages.

To organize the dataset in the same way as SQuAD dataset, (*paragraph*, *question*, *answer position*) triplets, we trace the passage in data split-1 in the original paragraph and re-compute the answer position as well. We finally make a synthetic data with about 50k questions and train the machine comprehension system only with our synthetic data.

As shown in Table 3.4, the machine comprehension system achieves EM/F1 score of 22.72/31.58 in public SQuAD dev set. This result is far below the result 68.78/78.56 of the case when the model is trained with the original training set. However, considering our synthetic data only consists of target answers with single named entity, we further check EM/F1 score of partial dev set that only has a single named entity as the answer. We find that in the 10k dev set, about 40 percent of the data has an answer with a single named entity and the machine comprehension system achieves EM/F1 score of 49.09/56.57 with those parts of the data. Since the SQuAD dataset is a human-made dataset, this result sufficiently shows that our Answer-Separated Seq2Seq can generate valid questions that can be acceptable both by human and machine comprehension systems.

3.7 Conclusion

In this chapter, we investigate the advantages of answer separation in neural question generation. We observe that existing NQG models suffer from a serious problem: a

significant proportion of generated questions include words in the question target, resulting in the generation of unintended questions. To overcome this problem, we introduce a novel NQG architecture that treats the passage and the target answer separately to better utilize the information from the both sides. Experimental results show that our model has a strong ability to generate the right question for the target answer in the passage. As a result, it yields a substantial improvement over previous state-of-the-art models.

Chapter 4

SEQUENCE-LEVEL TRAINING OF UNCONDITIONAL TEXT GENERATION

Generating realistic text is an important task with a wide range of real-world applications, such as machine translation [19], dialogue generation [23], image captioning [81], and summarization [82]. A language model is the most common approach for text generation, and it is typically trained via maximum likelihood estimation (MLE), specifically in an autoregressive fashion.

Although MLE-based methods have achieved great success in text generation, there are two fundamental issues that call for further research. The first problem is that MLE suffers from *exposure bias* [51]: during training, the model sequentially generates the next word depending on the ground-truth words; however, the model relies on its previously generated words at inference time. Therefore, the cumulative effect of incorrect predictions in the text sequence results in low-quality samples. The second problem lies in that the objective functions of MLE-based methods are rigorous [52]; the models are forced to learn every word in the target sentence. Under this strict guidance, the ability of language models to generate diverse samples can be severely limited.

In recent years, Generative Adversarial Networks (GANs) [54] have drawn atten-

tion as a remedy to the above problems. However, applying GANs to text-related tasks is challenging due to the discrete nature of text. In the inference phase of the text generation, the model iteratively samples the next word from the distribution of vocabulary. As this step includes the sampling process that hinders the backpropagation of the gradients from the discriminator, several approximation methods have been proposed to avoid the non-differentiability issue [55, 56, 17, 57].

Depending on the data space in which the GAN's discriminator operates, text GANs can be categorized into two groups: continuous-space methods and discrete-space methods. For discrete spaces, one prominent research line adopts the reinforcement learning (RL) technique to address the non-differentiability issue directly [55, 56, 83]. In the RL setting, GANs treat the generator as a stochastic policy to synthesize realistic samples. The generator is optimized via policy gradient methods by incorporating the reward signals from the discriminator. These signals are computed from a complete sequence rather than individual words in text. This RL approach can consider the final form of the text, thus it resolves the discrepancy between the training and inference stages in the MLE method. However, it has significant limitations, such as an excessive dependency on MLE pretraining, and severe mode collapse [84].

Other methods employ approximation techniques to transform discrete text into continuous representation. Such approaches include substituting next-word sampling in the generation phase with continuous relaxation [85, 86] or adopting an autoencoder architecture to learn an implicit data manifold in a continuous space instead of directly modeling the discrete text [17, 87]. In these approaches, the discriminator distinguishes between the synthetic and real text representations in the continuous space. As the discriminator only learns to distinguish an approximated representation of text, these approaches cannot provide direct feedback concerning the entire text's correctness.

In this work, we propose a novel text GAN architecture, called ConcreteGAN, which promotes the collaborative training of the ***cont***inuous-space and dis***crete***-space methods. Specifically, in the continuous space, a latent code representation of the

synthetic text is learned jointly with an autoencoder. Then, the textual output generated from the latent code is further updated via RL training. In this way, Concrete-GAN simultaneously regularizes the text generation process within the continuous and discrete data spaces. The interplay between adversarial trainings in the two data spaces takes the following advantages; 1) it reduces RL training variance through the regularization of the latent code representation; and 2) it alleviates exposure bias in continuous-space methods. To the best of our knowledge, our proposed method is the first work to train a text GAN combining both continuous-space and discrete-space methods.

We evaluate our model on three benchmark datasets: the COCO Image Caption corpus, the Stanford Natural Language Inference corpus, and the EMNLP 2017 WMT News corpus. Extensive experiments show that our model surpasses the existing text GAN models and achieves a substantial improvement with respect to quality, diversity, and global consistency. In addition, we provide comprehensive analyses of the latent code space. Compared to the GANs that work only in a continuous space, the synthetic code space generated by our model is more similar to the latent code space of real text. This behavior demonstrates that the proposed approach effectively regularizes the latent code space, which helps to reduce the variance of RL training.

4.1 Background

In this section, we first give a brief description of GANs. Then we introduce two lines of research for text generation, including continuous-space methods and discrete-space methods.

4.1.1 Generative Adversarial Networks

GANs are one of the implicit generative models that do not require a tractable likelihood function. Thus, they can be applied to practical situations such as imitating the

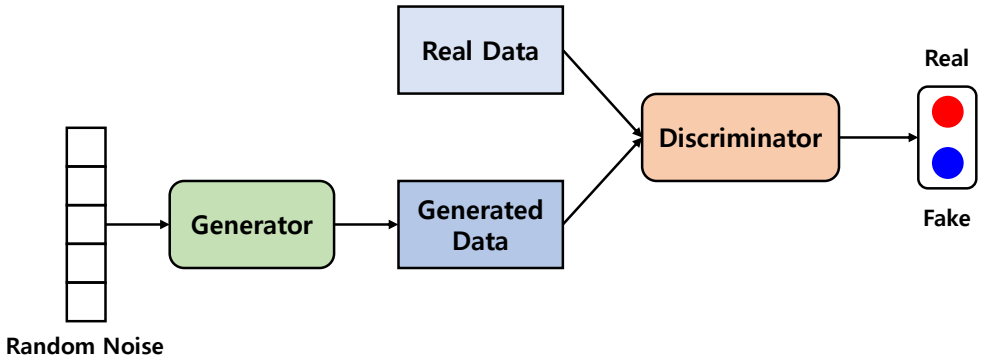

Figure 4.1: Illustration of structure of a generative adversarial network (GAN). GAN is basically composed of a generator which tries to generate real-looking samples, and a discriminator which aims at distinguishing generated samples from the real data.

distribution of high-dimensional complex data.

In general, GANs have a generator and a discriminator as their basic components, as shown in Figure 4.1. The generator tries to imitate the real data distribution, and the discriminator tries to distinguish generated samples from real data. The iterative interplay between these two components improves their strength against each other and provides significant performance enhancement in each of them. One can formulate the objective function of the GAN as a minimax game:

$$\min_G \max_D F(D,G) = \mathbb{E}_{\mathbf{x}\sim data}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim noise}[\log(1 - D(G(\mathbf{x})))], \tag{4.1}$$

where G and D are the functions for the generator and discriminator, respectively.

GANs have shown significant achievements in various deep learning applications, especially in computer vision research [88, 89, 90]. However, when applied to text generation, GANs suffer from the non-differentiability issue due to the discrete nature of text. Recently, various methods have been proposed to circumvent this issue, which can be broadly classified into two categories: continuous-space methods and discrete-space methods.

4.1.2 Continuous-space Methods

Several studies on GANs sidestep the non-differentiability by reformulating the learning objective in a continuous space. The adversarially regularized autoencoder (ARAE) [17], as a representative model, employs an autoencoder to learn an implicit data manifold, mapping discrete text into a continuous latent representation. In this model, the encoder and the generator are trained adversarially. Based on the ARAE, LATEXT-GAN [87] uses an additional approximated representation of text called soft-text, which is the reconstructed output of the autoencoder. Then, they employs two discriminators for each approximated representation in the continuous spaces.

Another line of work exploits a differentiable continuous relaxation, (i.e., Gumbel-softmax [85, 91] or soft-argmax [86]), to replace next-word sampling in the generation phase.

4.1.3 Discrete-space Methods

SeqGAN [55] is the first work addressing the non-differentiablity issue within a discrete space by introducing an RL technique into GAN training. Specifically, this approach considers the generated words as the current state and the generation of the next word as an action. In this scenario, the generator is optimized via a policy gradient, where the reward is computed by the discriminator through a Monte Carlo search. MaliGAN [92] proposes a normalized maximum likelihood objective. Combined with several reduction techniques, it reduces the variance of the reinforcement learning rewards and the instability of the GAN training dynamics. LeakGAN [93] devises a hierarchical architecture for the generator to address the sparsity issue in the long text generation. The generator is guided by the latent feature leaked from the discriminator at all generation steps. RankGAN [56] relaxes the binary restriction of the discriminator by exploiting relative ranking information between the real sentences and generated ones. This increases the diversity and richness of the sentences. All of the above methods use the maximum likelihood pretraining, followed by small amounts of adversarial

fine-tuning. ScratchGAN [83] first achieved a performance comparable to that of MLE methods without any pretraining.

4.2 ConcreteGAN

In this work, we propose a novel text GAN architecture that promotes the collaboration of two adversarial trainings in a continuous space and a discrete space, respectively. We adopt the alternating training of the two methods in each iteration, rather than MLE pretraining commonly used for discrete-space methods. The architecture of our model is shown in Figure 4.2.

Our model consists of the following four components: (1) RNN-based autoencoder is composed of an encoder and a decoder. The encoder provides a latent code representation of real text in a continuous space. The decoder, as a text-generator, yields textual outputs by interpreting the latent code from encoder or code-generator. (2) Code-generator maps a random noise to a latent code representation with the goal to imitate the distribution of the encoder. (3) Code-discriminator is the code-generator's opponent, and is adversarially trained to distinguish between latent codes from encoder and code-generator. (4) Text-discriminator evaluates complete sequences from real data distribution or decoder(or text-generator) distribution. The computed scores are used as the rewards to train the decoder via the policy gradient algorithm. The interplay between two adversarial trainings has a complementary effect, improving both the quality and the diversity of generated text.

4.2.1 Autoencoder Reconstruction

Let $\mathbf{x} \in \mathbf{X}$ be the input sequence and $\mathbf{z} \in \mathbf{Z}$ be the latent code of an autoencoder. We use a conventional RNN autoencoder that consists of two parts: an encoder network and a decoder network. The encoder network $f_\phi : \mathbf{X} \mapsto \mathbf{Z}$ (parameterized by ϕ) maps the input sequence $\mathbf{x}$ to the latent code $\mathbf{z}$, which is represented as the last hidden state.

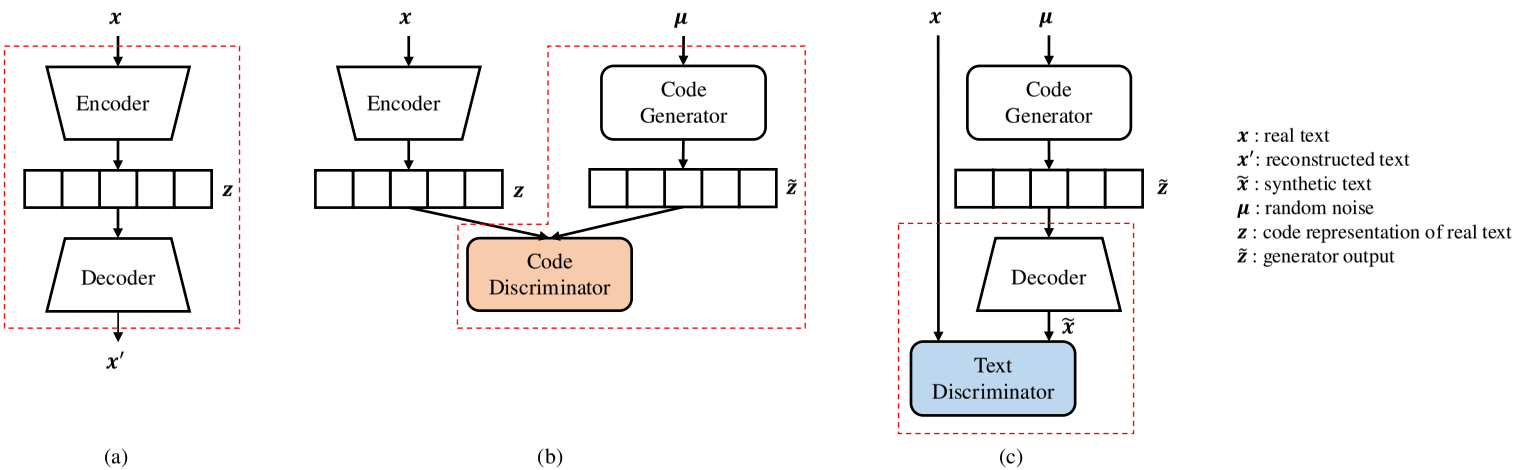

Figure 4.2: The overall architecture of the proposed model, which is composed of an autoencoder, a code-generator, a code-discriminator and a text-discriminator. Each training iteration of our model has three steps: (a) autoencoder reconstruction, (b) adversarial training of code-generator and code-discriminator in a continuous space, and (c) adversarial training of decoder and text-discriminator in a discrete space. The red dotted lines specify the modules to be updated in each step. The collaborative interplay between two adversarial training in continuous (latent code) and discrete (natural text) space regularizes the text representations in different spaces [4].

The decoder function $f_\psi : \mathbf{Z} \mapsto \mathbf{X}$ (parameterized by ψ) reconstructs the original input $\mathbf{x}$ conditioned on the encoded latent code $\mathbf{z}$. Here, we use a gated recurrent unit (GRU) for both the encoder and decoder networks, whose parameters are trained using the cross-entropy loss function:

$$\mathbf{z} = f_\phi(\mathbf{x}) \tag{4.2}$$

$$\min_{\phi,\psi} L_{rec}(\phi, \psi) = -\mathbb{E}[\log p_\psi(\mathbf{x}|\mathbf{z})] \tag{4.3}$$

4.2.2 Adversarial Training in the Latent Code Space

The next step of autoencoder reconstruction is the adversarial training of code-generator $G_\theta(\boldsymbol{\mu})$ and code-discriminator $D_\omega(\mathbf{z})$. The code-generator aims to imitate the distribution of real text in the continuous latent code space that is represented as the last hidden state of the encoder. Given a random noise vector $\boldsymbol{\mu}$ from a fixed distribution, such as a standard Gaussian distribution, the code-generator $G_\theta(\boldsymbol{\mu})$ outputs a vector $\tilde{\mathbf{z}}$ that has the same shape as the last hidden state of the encoder. On the other hand, the code-discriminator $D_\omega(\mathbf{z})$ learns to distinguish code-generator's output from the latent representations of real text. We use multilayer perceptrons (MLPs) with residual connections for both $G_\theta(\boldsymbol{\mu})$ and $D_\omega(\mathbf{z})$. Since WGAN, a variant of GAN which uses the 1-Wasserstein distance to measure the difference between the model and target distributions, showed better performance in terms of training stability and reduction of mode collapse, we adopt WGAN with gradient penalty (WGAN-GP) for optimization.

$$\tilde{\mathbf{z}} = G_\theta(\boldsymbol{\mu}) \tag{4.4}$$

$$\begin{aligned} \min_{\omega} L_\omega = &\mathop{\mathbb{E}}_{\tilde{\mathbf{z}} \sim \mathbb{P}_G}[D_\omega(\tilde{\mathbf{z}})] - \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathbb{P}_{f_\phi}}[D_\omega(\mathbf{z})] \\ &+ \lambda \mathop{\mathbb{E}}_{\hat{\mathbf{z}} \sim \mathbb{P}_{\hat{\mathbf{z}}}}[(\|\nabla_{\hat{\mathbf{z}}} D_\omega(\hat{\mathbf{z}})\|_2 - 1)^2] \end{aligned} \tag{4.5}$$

$$\min_{\theta} L_\theta = -\mathop{\mathbb{E}}_{\tilde{\mathbf{z}} \sim \mathbb{P}_G}[D_\omega(\tilde{\mathbf{z}})], \tag{4.6}$$

where $\hat{\mathbf{z}} = t\tilde{\mathbf{z}} + (1-t)\mathbf{z}$ with $0 \leq t \leq 1$

4.2.3 Adversarial Training with Textual Outputs

Along with the adversarial training in the latent code space, we build another adversarial training loop that operates on the discrete textual outputs.

Given a code-generator with fixed weights, we model the decoder, which yields the textual outputs, as a policy and apply policy gradient method to optimize it. The text-discriminator D_ρ is utilized to evaluate the generated sequence and provide the reward R_t. Following previous works, we use REINFORCE [94], a Monte Carlo (MC) variant of the policy gradient algorithm, for gradient estimation of the decoder training.

Since the reward signal can be calculated only when the entire sequence is completely generated, several approximation methods are proposed to obtain an intermediate reward for each generated token. While an MC search with a roll-out policy [55] is the method adopted in most research, it is computationally expensive even with a feed-forward discriminator. From our preliminary experiments, we find that GRU-based sequential discriminator shows better performance than a CNN discriminator with MC search in terms of computation time and evaluation results. With this empirical intuition, we use GRU-based discriminator as follows:

$$\begin{aligned}\min_{\rho} L_\rho = &-\sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}\sim data}[\log D_\rho(x_t|x_1,...,x_{t-1})] \\ &+\sum_{t=1}^{T'} \mathbb{E}_{\tilde{\mathbf{x}}\sim f_\psi(\tilde{z})}[\log D_\rho(\tilde{x}_t|\tilde{x}_1,...,\tilde{x}_{t-1})]\end{aligned} \tag{4.7}$$

$$R_t = \sum_{s=t}^{T} \gamma^{s-t} D_\rho(\tilde{x}_t|\tilde{x}_{t-1},...,\tilde{x}_1) \tag{4.8}$$

$$\nabla_\psi = \sum_{t=1}^{T}\sum_{n=1}^{N} \nabla_\psi \log f_\psi(\tilde{x}_t|\tilde{x}_1,...,\tilde{x}_{t-1},\mathbf{z}) R_t, \tag{4.9}$$

where γ is a discount factor such that $0 < \gamma < 1$ and N is the size of the mini-batch. The overall learning procedure is shown in Algorithm 1.

As a result of adversarial training in a continuous space, code-generator can provide an regularized latent representation of text sequence. This leads to the effective

Algorithm 1 ConcreteGAN Training

Require: text encoder f_ϕ; shared decoder f_ψ; code-generator G_θ; code-discriminator D_ω; text-discriminator D_ρ; real text data $\mathbf{x} \in \mathbf{X}$; random noise vector $\boldsymbol{\mu}$;

for each training iteration **do**

(1) Train the autoencoder for reconstruction (f_ϕ, f_ψ)

Compute $\mathbf{x}' = f_\psi(f_\phi(\mathbf{x}))$

Backprop loss $L_{rec}(\mathbf{x}, \mathbf{x}')$

(2) Adversarial training in the code space (G_θ, D_ω)

Compute $\mathbf{z} = f_\phi(\mathbf{x})$ and $\tilde{\mathbf{z}} = G_\theta(\boldsymbol{\mu})$

Backprop loss $L_\omega(\mathbf{z}, \tilde{\mathbf{z}})$ to update D_ω

Backprop loss $L_\theta(\tilde{\mathbf{z}})$ to update G_θ

(3) Adversarial training with textual output (f_ψ, D_ρ)

Compute $\tilde{\mathbf{x}} = f_\psi(G_\theta(\boldsymbol{\mu}))$

Backprop loss $L_\rho(\mathbf{x}, \tilde{\mathbf{x}})$

Train f_ψ via policy gradient ∇_ψ

end for

restriction on the search space of the RL-policy decoder, acting as a guideline for generating a sentence within bounded space. In adversarial training in a discrete space, the decoder learns to better capture the structure of text, such as a phrase, rather than the choice of words. This process contributes to mitigate the exposure bias of autoencoder, which further affects the training process of continuous space.

4.3 Experiments

To demonstrate the efficacy of our proposed method, we evaluate our model on various real-world datasets. In what follows, we give a detailed description of the whole evaluation process, from the experimental settings to the experimental results. We provide a performance comparison with state-of-the-art models as well as several analyses on

Table 4.1: Statistics of the standard benchmark datasets for evaluating text GANs

	COCO	SNLI	EMNLP
Vocabulary size	4,682	42,423	5,255
Sequence length	<37	<81	<51
# of sentences (train)	10k	701k	270k
# of sentences (dev)	10k	13k	10k
# of sentences (test)	10k	13k	10k

the code space.

4.3.1 Dataset

We carry out experiments on three standard benchmark datasets for evaluating text GANs: COCO Image Caption (COCO) dataset [79], Stanford Natural Language Inference (SNLI) corpus [95] and EMNLP 2017 WMT News (EMNLP) dataset. The statistics of each dataset are presented in Table 4.1.

For the SNLI dataset, considering the data distribution, we set a maximum sentence length of 15 and a vocabulary size of 11k. Each dataset represents different experimental environments, which have a critical impact on the unsupervised training of the text generation model: COCO for small-sized data with short text, SNLI for big-sized data with short text, and EMNLP for mid-sized data with long text.

4.3.2 Experimental Settings

We implement our model using TensorFlow 1.15 and train the model with up to 200,000 iterations. For all experiments, we use the same model, loss function, and hyperparameters across the set of datasets, but different vocabulary sizes.

Autoencoder

The autoencoder is made up of an encoder GRU and a decoder GRU with 300 hidden units. We use 300-dimensional GloVe word embeddings trained on 840 billion tokens to initialize both the encoder and the decoder, and they are fine-tuned separately during training. The encoder output is normalized with l2-normalization. The input to the decoder is augmented by the output of the previous time step with a residual connection at every decoding time step. Additive Gaussian noise is injected into the encoder output and decays with a factor of 0.995 every 100 iterations. We use ADAM [78] optimizer with an initial learning rate of $1e^{-03}$. Gradient clipping is applied if the norm of gradients exceed 5.

Generator & Discriminators

The code-generator and the code-discriminator are 2-layer 300-dimensional MLPs with residual connections between each layer. We use a Leaky ReLU for the activation function. The text-discriminator is a 1-layer GRU with 300 hidden units that has the same structure as the decoder. We use ADAM [78] optimizers and set the initial learning rate of the code-generator and two discriminators as $5e^{-06}$ and $5e^{-03}$ respectively. Gradient clipping is applied to the text-discriminator if the norm of gradients exceed 5.

4.3.3 Evaluation Metrics

The evaluation of natural language generation models is difficult since there is no single metric to measure the quality of various features of the language. In general, there are two aspects of natural language to be evaluated: quality and diversity.

BLEU & Backward BLEU

Following previous works, we use BLEU score [96] as the metric of quality. For each dataset, we first sample the same amount of generated text as the held-out test data. Then, for each generated text, the corpus-level BLEU score is calculated with the entire test data as a reference data [97]. Reference [98] have proposed to use backward BLEU (B-BLEU) for the measurement of diversity. For the B-BLEU score, we evaluate each of the test data given the entire generated data as a reference. Intuitively, the BLEU score measures the precision of generated text, while the B-BLEU score measures the recall of generated text. For both scores, a larger value indicates better performance.

Fréchet distance

Reference [99] proposed an automatic evaluation metric called the Fréchet InferSent Distance (FD), which evaluates the outputs of text generation models. The FD calculates the Fréchet distance between real text and generated text in the pretrained embedding space. This metric can capture both quality and diversity along with the global consistency of the text. Since the metric is known to be robust to the embedding model, as suggested in [83], we use Universal Sentence Encoder [100] to compute the sentence embedding of texts for our experiments.

4.3.4 Experimental Results for Quality & Diversity

We compare our model with an MLE baseline along with other state-of-the-art text GANs, such as SeqGan [55], RankGan [56], MaliGan [92], and ScratchGan [83] based on Texygen [97], which is an evaluation platform for text GANs. The MLE baseline is an RNN with MLE objective which has the same structure as the decoder of the proposed model. In addition, we detach the text-discriminator from our model and train the remaining part with the same training strategy as for the ARAE [17], which is one of the most representative continuous-space text GANs. Our RL-detached model shows superior performance over the original ARAE model (detailed information is

provided in Appendix 7.2. We call the model ARAE* in the following sections. Every score is averaged over five runs, and they have a standard deviation smaller than 0.005.

Table 4.2 reports BLEU and B-BLEU scores of text GANs trained on the EMNLP dataset. While recently proposed ScratchGan surpasses the previous state-of-the-art text GANs by a significant margin, ConcreteGAN shows superior performance over ScratchGan. Interestingly, our implementation of an ARAE-like model (see "ARAE*" in the table) performs better than most of the discrete-space methods. The effect of our model stands out in larger n-grams, which means that commonly-used combinations of words, such as phrases, can be generated with better quality and diversity.

Then, we compare the performance on another commonly used corpus, which is a part of the original COCO image caption dataset, and has a very small amount of training data. As shown in Table 4.3, ConcreteGAN performs better than most of the discrete-space methods in generating longer combinations of words. However, we find that all of the BLEU and B-BLEU scores of ARAE* are higher than those of discrete-space methods, including the proposed model. We conjecture that the lack of training data (i.e., 10k samples) cannot provide enough guidance for the RNN-based RL discriminator.

To see the effect on the dataset size, we conduct an additional experiment on the SNLI dataset, which is composed of a large amount of data with short sentences(i.e., 701k samples). In addition to the MLE baseline, We choose ScratchGAN, ARAE*, and ConcreteGAN, which represent the discrete-space methods, continuous-space methods, and combined approaches respectively, for comparison. Table 4.4 shows the BLEU and B-BLEU scores of these three models. With a large data for training the models, our proposed method surpasses ARAE* and achieves the best performance compared to other text GANs and the MLE baseline.

Table 4.2: Comparison of the generated sentence quality and diversity in terms of the corpus-level BLEU and B-BLEU scores on the EMNLP 2017 WMT News dataset. A larger value indicates better quality / diversity

Metrics	MLE	SeqGan	RankGan	MaliGan	ScratchGan	ARAE*	Ours
BLEU-2	0.843	0.761	0.736	0.764	0.830	0.824	**0.858**
BLEU-3	0.573	0.463	0.441	0.468	0.552	0.614	**0.658**
BLEU-4	0.328	0.228	0.204	0.231	0.309	0.391	**0.431**
BLEU-5	0.182	0.115	0.095	0.113	0.172	0.228	**0.257**
B-BLEU-2	**0.842**	0.693	0.671	0.684	0.832	0.808	0.806
B-BLEU-3	**0.571**	0.413	0.373	0.391	0.559	0.552	0.555
B-BLEU-4	0.328	0.216	0.191	0.197	0.317	0.330	**0.338**
B-BLEU-5	0.189	0.112	0.096	0.094	0.176	0.189	**0.197**

Table 4.3: Comparison of the generated sentence quality and diversity in terms of the corpus-level BLEU and B-BLEU scores on the COCO caption dataset. A larger value indicates the better quality / diversity

Metrics	MLE	SeqGan	RankGan	MaliGan	ScratchGan	ARAE*	Ours
BLEU-2	0.745	0.748	0.727	0.733	**0.762**	0.760	0.729
BLEU-3	0.509	0.514	0.491	0.497	0.535	**0.557**	0.526
BLEU-4	0.317	0.307	0.291	0.295	0.344	**0.374**	0.347
BLEU-5	0.196	0.187	0.175	0.178	0.221	**0.243**	0.222
B-BLEU-2	**0.775**	0.748	0.727	0.733	0.762	0.766	0.756
B-BLEU-3	**0.552**	0.514	0.491	0.497	0.535	0.547	0.536
B-BLEU-4	0.342	0.307	0.291	0.295	0.344	**0.359**	0.351
B-BLEU-5	0.219	0.187	0.175	0.178	0.221	**0.228**	0.226

Table 4.4: Comparison of the quality and diversity of sentences generated by state-of-the-art text GANs with BLEU and B-BLEU scores on the Stanford Natural Language Inference dataset. A larger value indicates the better quality / diversity

Metrics	MLE	ScratchGan	ARAE*	Ours
BLEU-2	0.843	0.814	0.848	**0.871**
BLEU-3	0.641	0.604	0.639	**0.681**
BLEU-4	0.440	0.410	0.422	**0.466**
BLEU-5	0.307	0.277	0.272	**0.311**
B-BLEU-2	**0.831**	0.808	0.821	0.817
B-BLEU-3	0.634	0.594	0.635	**0.636**
B-BLEU-4	0.439	0.400	0.435	**0.446**
B-BLEU-5	0.290	0.269	0.288	**0.301**

4.3.5 Experimental Results for FD score

We compare FD score between the real text distribution and the generated text distribution in the Universal Sentence Embedding space. Table 4.5 shows the FD score of each state-of-the-art model with different learning paradigm. Analogous to the results in the *Experimental results for Quality & Diversity Evaluation*, the FD scores of all three models on the COCO corpus are fairly high. We explain this result as a natural outcome of the lack of training data. In other corpora with large training data, our model shows the best performance, which means that it can generate text that has the most similar distribution to the real text.

4.3.6 Human Evaluation

We further conduct a human evaluation for textual sample quality of ConcreteGAN and other methods. Following previous work [91], we randomly sample 100 sentences from each model and ask ten different people to score each sample on Amazon Mechanical Turk. We provide detailed criteria of human evaluation in Appendix 7.3. As

Table 4.5: Comparison of the Fréchet distance between the real text distribution and generated text distribution in the Universal sentence embedding space. A lower value indicates the a smaller distance between the two distributions.

Dataset	MLE	ScratchGan	ARAE*	Ours
COCO	0.245 (±0.003)	0.259 (±0.005)	0.238 (±0.004)	**0.235** (±0.004)
SNLI	0.010 (±0.002)	0.011 (±0.001)	0.011 (±0.001)	**0.008** (±0.001)
EMNLP	0.021 (±0.002)	0.021 (±0.001)	0.022 (±0.001)	**0.019** (±0.001)

Table 4.6: Human evaluation score of state-of-the-art models on the EMNLP 2017 WMT News dataset. One hundred random samples generated by each model are evaluated by 10 English native workers on Amazon Mechanical Turk.

Methods	MLE	ScratchGan	ARAE*	Ours
Human score	3.157 (±1.043)	3.157 (±1.048)	3.204 (±1.040)	**3.337** (±0.946)

shown in Table 4.6, the samples from ConcreteGAN are rated with the highest score compared to the state-of-the-art models of continuous-space and discrete-space models. Along with the experimental results in the previous sections, the human evaluation further demonstrate that the proposed method can generates human-like samples better than other methods.

4.3.7 Analyses of Code Space

In the previous section, the textual outputs of various text GANs are compared with diversified measurements. To demonstrate the effectiveness of the collaborative adversarial training in both the continuous and discrete spaces, we further analyze the behav-

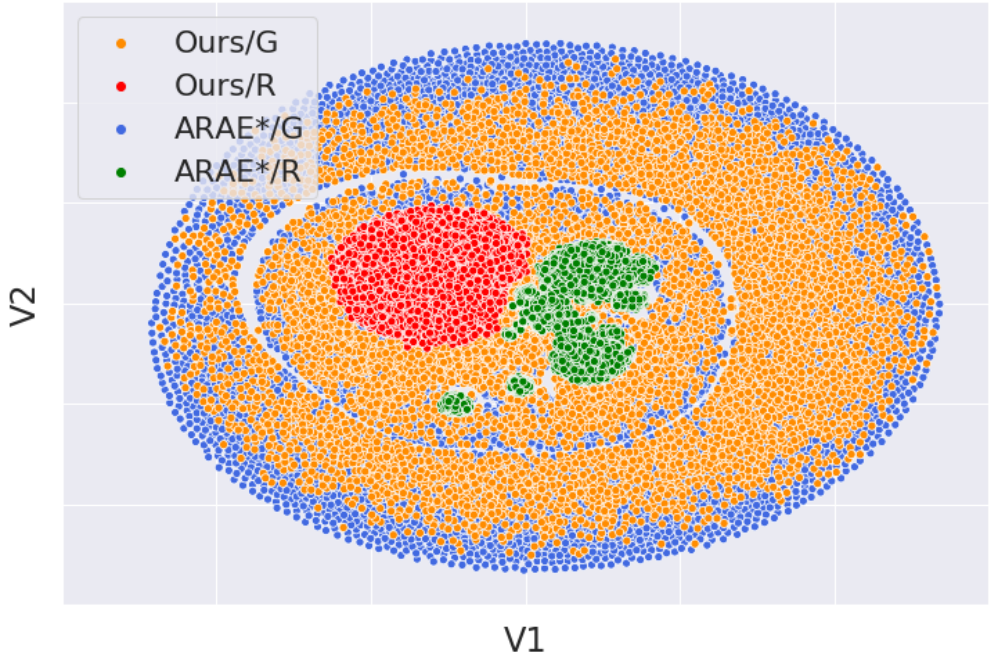

Figure 4.3: Comparison of latent code distribution via t-distributed stochastic neighbor embedding (t-SNE). G indicates the output distribution of the code-generator. R indicates the latent code distribution of the real text generated by the encoder [4].

ior of code-generators. As the goal of code-generator is to imitate the real distribution of text in the latent code space, we compare the outputs of the encoder and code-generator to examine the performance of code-generator. We independently gather the encoder's outputs from the real text inputs (i.e., the test dataset) and the code-generator's outputs from the random noise inputs.

Analysis on t-SNE Space

We first visualize the code distribution with t-SNE [101] for in-depth analysis. Figure 4.3 shows the t-SNE plots of four different latent code distribution; each of them represents the code-generator outputs of ConcreteGAN (Ours/G), the encoder outputs of ConcreteGAN (Ours/R), the code-generator outputs of ARAE* (ARAE*/G), and

Table 4.7: The Fréchet distance between the latent code distribution of real and synthetic text. The lower the value is, the closer the two distributions.

FD	SNLI	EMNLP
ARAE*	24.7	18.9
Ours	**15.5**	**16.2**

the encoder outputs of ARAE* (ARAE*/R). We see the encoders of both models map the real text to more compact latent spaces than the code-generators. Furthermore, the latent code space generated by our code-generator is more compact than that of ARAE* in the same embedding space. Considering that the two models (i.e., ConcreteGAN and ARAE*) employ the same encoder architecture, this result demonstrates that ConcreteGAN produces a compact and dense latent code distribution, which is more similar to the latent code space of real text.

Fréchet Distance between Latent Code Distribution

We further compare the Fréchet distance between the latent code distribution of real and synthetic text. Latent codes are obtained from the encoder and code-generator, respectively. As the codes are represented as embedding vectors, no external model for computing the sentence-embedding is required. As shown in Table 4.7, ConcreteGAN shows reduced Fréchet distance compared to ARAE* in both datasets: 24.7 to 15.15 in SNLI and 18.9 to 16.2 in EMNLP. These results demonstrate our model's superiority in generating latent codes compared to the previous baseline with a significant margin.

While the proposed model shows better performance in imitating the encoder than ARAE*, we see that the gap of Fréchet distance is smaller in the SNLI dataset than in the EMNLP dataset. The average length of a sentence in EMNLP is approximately three times larger than that of SNLI, and it is more difficult for the model to encode lengthy text to a fixed-size code representation. This observation calls for future research investigating the use of a different architecture (i.e., a CNN or Transformer) for

the encoder part.

4.4 Conclusion

In this chapter, we propose ConcreteGAN, a novel GAN architecture for text generation. Unlike previous approaches, ConcreteGAN promotes the collaborative training of the continuous-space and discrete-space methods. The interplay between two adversarial trainings has a complementary effect on text generation. From a continuous-space method, our model effectively reduces the search space of RL-policy decoder. Meanwhile, discrete-space training enables the model to capture the structure of text and thereby alleviate the exposure bias, which is caused by continuous-space methods. The experimental results on three standard benchmark datasets show that ConcreteGANs outperforms state-of-the-art text GANs in terms of quality, diversity, and global consistency.

Chapter 5

SEQUENCE-LEVEL TRAINING OF CONDITIONAL TEXT GENERATION

5.1 Introduction

The imitation of human intelligence is one of the grand goals of building AI systems. Until now, the most significant difference between the computer and the human has been creativeness. The origin of human creativeness can be answered easily: we learn through our whole life. On the other hand, neural network-enhanced AI systems also learn but are only restricted to a piece of knowledge that the systems targets at. In general, deep neural networks require tons of labeled data to do the target task near-perfectly. However, in real life, we can not provide complete data for training the model because of the variety of target tasks and the enormous cost of human labeling. As a remedy, several substitutes have been proposed, such as unsupervised training, semi-supervised training, and self-supervised training.

Specifically, in NLP, the most straightforward way to overcome the data issue is to use unlabeled data since they are everywhere with an unlimited amount. Some research then has been proposed to leverage linguistic information from unlabeled data. There are mainly two strategies that provide compelling evidence to benefit from utilizing

the unlabeled data for several NLP tasks: unsupervised feature-based approaches and unsupervised fine-tuning approaches. One typical example of unsupervised feature-based approaches is Embeddings from Language Models (ELMo) [102]. ELMo is a way of contextualized word embeddings learned from bidirectional language modeling, and it provides context-sensitive features. We can readily concatenate the ELMo embeddings to the existing model's embedding to enhance the performance.

Unsupervised fine-tuning approaches, such as BERT [5], require a minimal amount of task-specific parameters and share almost all of the network structure with the downstream tasks. Thus, simply replacing the language modeling head with task-specific classifiers and fine-tuning the entire network on the task-specific dataset enables quick and great adaption to the downstream task. Since those unsupervised approaches mostly depend on training a language model before adoption to the downstream tasks, we simply call them pre-trained language models.

The application of pre-trained language models enables AI systems to imitate humans' way of learning and building experience. Even when a considerable amount of labeled data is available, it is proved that learning good representations from unlabeled data actually promotes an effective performance boost. The rise of the pre-trained language model has significantly changed the paradigm of NLP research, expanding to other research areas as well.

In this chapter, we first investigate the application of a pre-trained language model on question generation. We take an unsupervised fine-tuning approach and exploit one of the most powerful pre-trained language models, T5 [7]. Following the fine-tuning method of original T5, we add a task-specific prefix "generate question" to the original input sequence before feeding it to the model. Simply token-level fine-tuned on the SQuAD 1.1 dataset, the model outperforms many state-of-the-art baselines. However, even based on the pre-trained language model, the autoregressive text generation model trained with MLE still suffers from exposure bias and token-level rigorous objective function. As a remedy, we further fine-tune the model with the sequence-level

objective function. As Chapter 4 suggests, text GAN approaches can evaluate the generated sentence as a whole and provide the text generation model feedback on the naturalness of the generated sentence. However, simple discrimination between the real and generated data may not provide enough information to train a target-specific model. To this end, we propose to use question-specific rewards to better optimize the model for the question generation problem. We define the reward as "answerable score" of the generated question, which is calculated by a BERT-based abstractive machine reading model called SpanBERT [60]. We provide two different definitions of the answerable score, each of which focuses on confidence and accuracy. Then, the question generation model is further optimized with a reinforcement learning (RL)-based algorithm called Proximal Policy Optimization (PPO).

The proposed methods are evaluated on SQuAD 1.1, a dataset originally designed for the extractive machine reading task. Since T5 fine-tuned in token-level already outperforms previous RNN-based state-of-the-art models, we mainly focus on the effect of sequence-level training. While quantitative analysis could not provide an overall assessment, we further provide qualitative analysis on the generated samples. Compared to the token-level training method, sequence-level training can update the model towards improving the pre-defined reward which assesses the generated question as a whole. This behavior demonstrates that sequence-level training of question generation can effectively improve generation quality.

5.2 Background

5.2.1 Pre-trained Language Model

A pre-trained language model is designed to be a black box that understands the human language so that it can be asked to do some specific task in the human language. Typically, a large amount of unlabeled data that can cover a broad range of characteristics of the language, such as Wikipedia dump data, is used to train the pre-trained

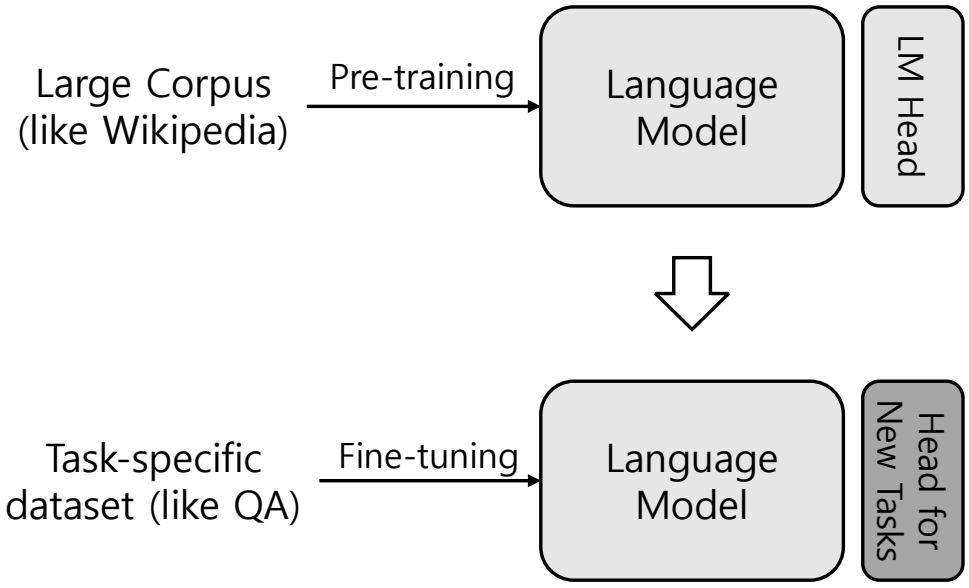

Figure 5.1: Usage of pre-trained language models. First, we use a large corpus (like Wikipedia) to train language modeling. Then, the model is fine-tuned on task-specific datasets.

language model. The adoption of the pre-trained language model includes two steps: pre-training and fine-tuning, as shown in Figure 5.1. During the pre-training step, the language model is first trained to learn the usage of various words and how the language is written in general. Then, in the fine-tuning phase, the model is fed a comparatively smaller and task-specific dataset, which is used to transfer the model parameter to the downstream tasks. In a nutshell, this procedure makes the model capable of performing downstream tasks.

Pre-trained language models have made colossal breakthroughs in almost all of the NLP tasks. The reason can be explained in one sentence: “They already understand the human language.” A model which is trained only on a task-specific dataset needs to understand both the language and the task only given limited resources (smaller dataset). On the other hand, the language model is capable of understanding the language since

it has already 'read' large language dumps during pre-training. Thus the language model can be easily fine-tuned to carry out the required task and achieves better performance than existing state-of-the-art models. In the following paragraph, we introduce some basic pre-trained language models [43, 45, 42, 103, 103, 104, 44, 105, 106, 107].

BERT

BERT [5], which is the abbreviation for Bidirectional Encoder Representations from Transformers, is a first-generation pre-trained language model presented by researchers at Google AI Language. BERT has fired up the research on the various pre-trained language models because it has brought innovation to the machine learning community by achieving state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1) [74], Natural Language Inference (MNLI) [108], and others. The main contribution of BERT is the application of the Transformer encoder in language modeling. The Transformer is an attention-based seq2seq model which is initially presented for neural machine translation. It is the first trial of the Transformer that the Attention Mechanism is used as the central architecture rather than an extra module. The adoption of the attention-based network to language modeling indicates an important characteristic: it enables bidirectional training, where the language model looks at a text sequence from both directions. This mechanism is quite different from conventional language modeling, which looks at a text sequence either from left to right or in a combined left-to-right and right-to-left manner. Experimental results show that a bidirectional language model has a a more profound sense of linguistic context and flow than unidirectional language models.

BERT is a denoising-based language model, which is also called a masked language model (MLM). The input to the BERT is masked word sequences, that a certain amount of words (15% in the paper) in each sequence are replaced with [MASK] tokens. Based on the contextual information provided by surrounding non-masked words, BERT is trained to predict the original words in the position of [MASK] to-

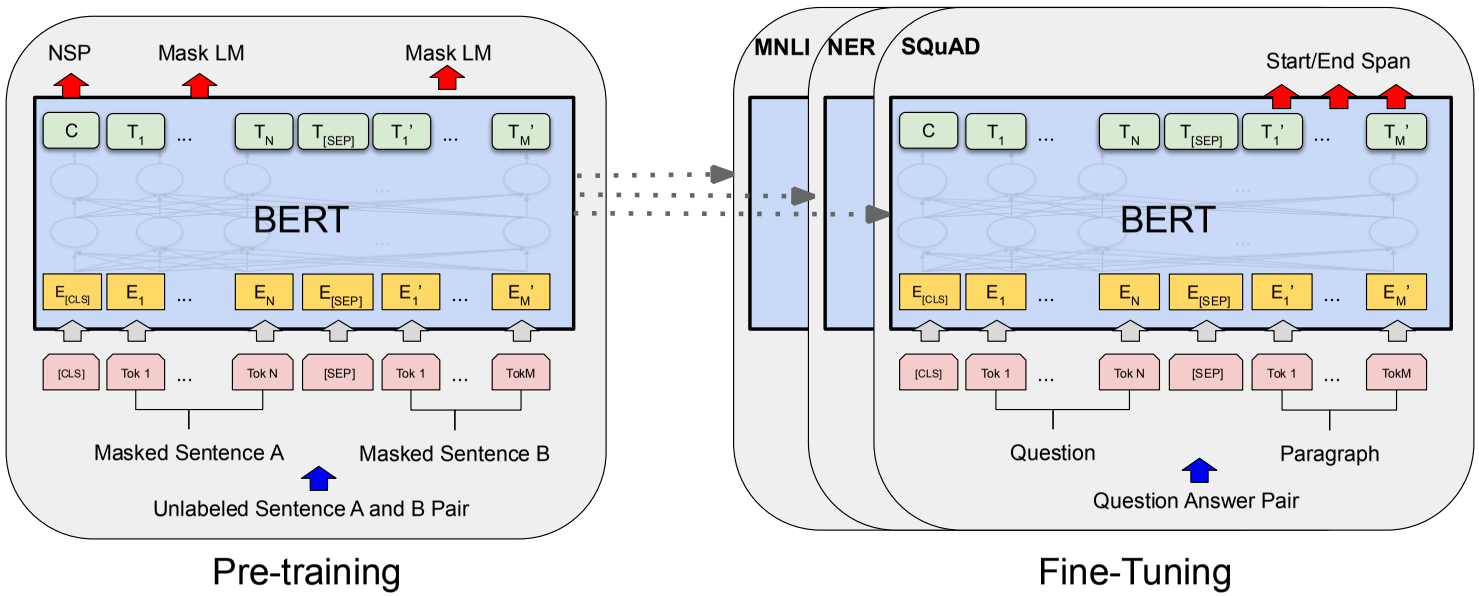

Figure 5.2: Overall pre-training and fine-tuning procedure for BERT [5].

ken. The objective function of BERT takes into consideration only the prediction of the masked values and ignores the predictions of the non-masked words. As a consequence, more training steps may be required to make the model converge while the improvements that BERT has deeper comprehension of the language outweigh the increased training cost.

BERT has another training objective: next sentence prediction (NSP). The model takes as input pairs of sentences and learns to classify if the second sentence in the pair is the subsequent sentence in the original document. For doing NSP, every first token of the BERT input is defined as a [CLS] token, and the corresponding output denotes the classification result. 50% of the inputs are consecutive pairs of sentences extracted from the original document, while the others are pairs of sentences that the second sentence is randomly selected. This procedure explicitly enables the modeling of understanding the relationship between sentences.

As shown in Figure 5.2, using BERT for downstream tasks is straightforward: substituting only the language model head to others or adding a small layer, BERT can be applied to a wide variety of NLP tasks.

- In the machine reading task (e.g., SQuAD v1.1), given a paragraph (context) and a question regarding it, a model is asked to determine the answer span inside the paragraph. Based on BERT, a machine reading model is trained by learning two different vectors representing positions of the beginning and the end of the answer in the paragraph.

- In sentence classification problems such as sentiment analysis, a model receives a text sequence and is required to classify it among several candidate labels. Simply adding a classification layer on top of the BERT output for the [CLS] token is the most basic form.

In addition to the traditional NLP tasks, BERT has expanded its range of application to other domains such as biomedical research [109], computer vision re-

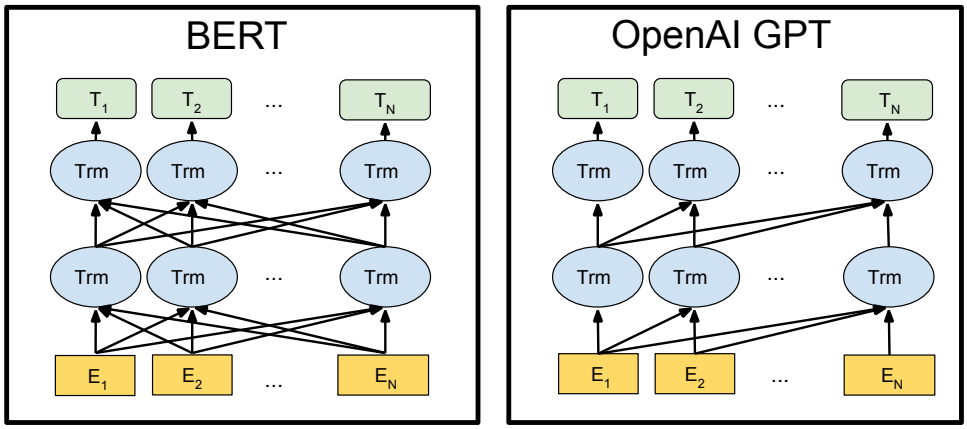

Figure 5.3: Structural difference between BERT and GPT-2 [5].

search [110].

GPT-2

[41] demonstrates that GPT-2, a pre-trained language model, can be used to solve downstream tasks without any direct supervision or modification on architecture. GPT-2, a Transformer-based language model with 1.5B-parameter, is trained on a large and diverse dataset called WebText, composed of text scraped from 45 million web links. The model achieves state-of-the-art performance on several language modeling datasets along with various downstream tasks. Figure 5.3 shows the main difference between the BERT and the GPT-2: BERT adopts only the encoder part of the Transformer, while the architecture of GPT-2 is very similar to the decoder-only Transformer.

XLNet

XLNet [6] is a generalized autoregressive pre-training method that enjoys the strengths of both autoregressive language modeling (e.g., Transformer-XL [111], and GPT-2) and auto-encoding language modeling (e.g., BERT) while avoiding their weakness.

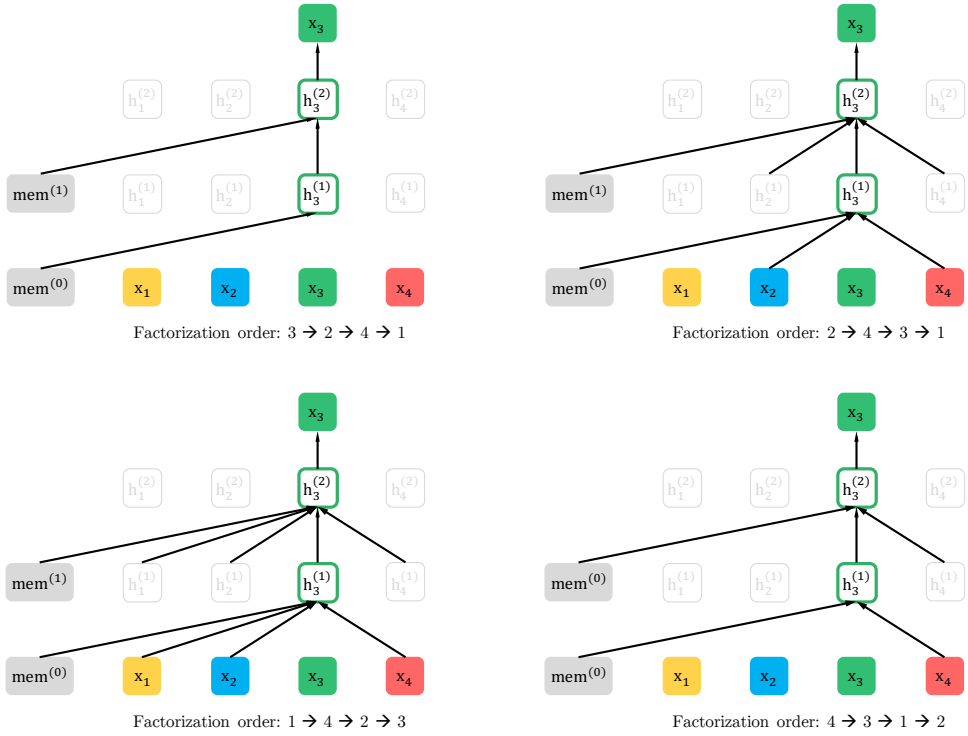

Figure 5.4: Illustration of the permutation language modeling of XLNet [6].

XLNet encodes the text sequence from both sides, just like BERT, which means it depends on a bidirectional context to predict the surrounded tokens. To this end, XLNet maximizes the expected log-likelihood of a sequence with respect to all possible permutations of the factorization order [6] as shown in Figure 5.4. Furthermore, as an autoregressive language model, XLNet is free from BERT's limitations resulting from token masking, such as the discrepancy between pre-training and fine-tuning and the assumption that unmasked tokens are independent of each other, and it does not rely on data corruption. The segment recurrence mechanism and relative encoding scheme of Transformer-XL are further integrated into XLNet to improve architectural designs for pre-training.

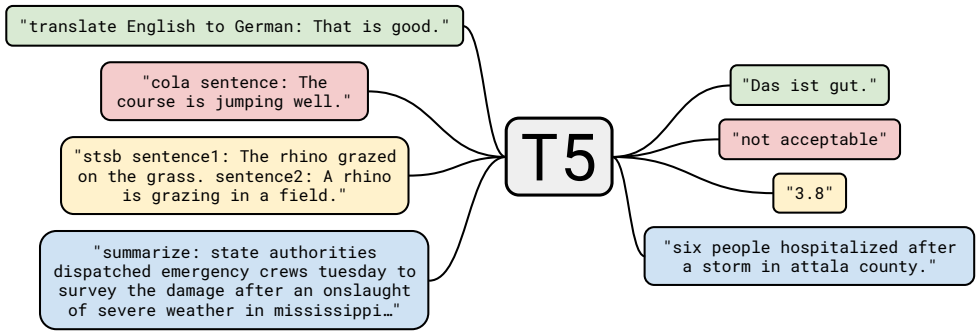

Figure 5.5: Illustration of text-to-text learning mechanism in T5 [7].

T5

Text-to-Text Transfer Transformer (T5) [7] is a unified approach to transfer learning in NLP that treats each NLP problem as a "text-to-text" problem. As the name denotes, the "Text-to-text" framework (Figure 5.5) lets the model understand what to do by the text itself without additional features. Specifically, a task-specific prefix is added to the original input sentence (e.g., "translate English to Korean:") to notify the model of the defined task. Such a framework does not require additional model, objective, training procedure, and decoding process for a wide variety of English-based NLP tasks, including document summarization, sentiment classification, question answering, and machine translation. Trained on the large corpus of web-crawled data called Colossal Clean Crawled Corpus (C4), T5 gets state-of-the-art results on a number of NLP tasks.

5.2.2 Proximal Policy Optimization

Policy gradient methods are a type of model-free RL technique, which is one of the driving forces to recent breakthroughs in deep neural networks for control problems, from video games, to 3D locomotion, to Go. However, despite being one of the most significant technical leaps, getting outstanding performance via policy gradient methods is challenging. For example, they are susceptible to step size: if too small, the

training process may cost much time; if too large, one might suffer from unstable learning, leading to policy crashing. In addition, due to the poor sample efficiency, it takes millions (or billions) of time-steps for many policy gradient algorithms to learn a simple task.

Several policy gradient variants, which constrain or optimize the size of a policy update, are proposed to overcome those limitations of the original policy gradient methods, such as Trust Region Policy Optimization (TRPO) [112] and Actor-Critic with Experience Replay (ACER) [113]. Though improved over the policy gradient algorithms, these methods still suffer from their trade-offs: For those algorithms that share parameters between a policy and a value function or auxiliary losses, TRPO is not easy to be applied. ACER is complex and fragile to implement, requiring additional code for a replay buffer and off-policy corrections.

Proximal Policy Optimization (PPO) [58] is a first-order approximation of TRPO, which enjoys much more straightforward implementation, better sample complexity, and better empirical performance. The main objective PPO algorithm tries to optimize is defined by the following equation:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{5.1}$$

where epsilon (ϵ = 0.2) is a hyperparameter, and $r_t(\theta)\hat{A}_t$ represents the "surrogate" objective function from TRPO. By adding a restriction on the surrogate objective with clipping mechanism, PPO prevents r_t from moving outside the interval $[1-\epsilon, 1+\epsilon$. The final objective is defined as a lower bound on the surrogate objective since the minimum value between the clipped and unclipped objective is taken. In the end, we can easily figure out that PPO's motivation is to keep the change in probability ratio if it improves the objective, restricting the harmful changes simultaneously.

5.3 Methods

The proposed method includes two phases of training: adapting a pre-trained language model to the question generation task with token-level training and further optimize the model with task-specific reward under sequence-level training. Specifically, a question-specific reward called "answerable score" is defined to encourage the model to generate more human-like questions.

5.3.1 Step One: Token-level Fine-tuning

The first part of the proposed method is to fine-tune the 60M-parameter version of the T5 model (T5-small) [7] which is already pre-trained on C4 dataset. The model architecture is just a sort of vanilla encoder-decoder transformer [2] with six layers, 512 hidden states, 2048 feed-forward hidden states, and eight heads. We use the checkpoint from Hugging Face Model Hub [114] to initialize weight parameters. We first perform supervised fine-tuning of T5 with SQuAD 1.1 [74], a dataset for machine reading which also can be used for question generation (Section 3.5.1). During the pre-training period of T5 model, a task-specific text "generate question: " is added to every input sequence explicitly for enabling a single T5 model to manage a diverse set of tasks simultaneously. This enables the model to share the whole weight parameters regardless of the task and further signal the model to generate the task-specific sequence in natural language form. In the same manner, we add a prefix "generate question:" to every encoder input sequence before feeding it to the model.

5.3.2 Step Two: Sequence-level Fine-tuning with Question-specific Reward

Since the token-level training methods for text generation models have several limitations (e.g., exposure bias and rigorous loss function), we investigate a training method that can evaluate the generated sequence as a whole. In particular, in this chapter, we

focus on a sequence-level training method for one of the conditional text generation problems, question generation. Following [59], we use the PPO algorithm, one of the high-performance and easy-to-use policy gradient methods, with a question-specific reward called answerable score.

First of all, we assume that the text generation procedure is a sequential decision-making process for applying RL. As mentioned in section 2.5, RL consists of some essential components such as action, state, and reward. We treat the fine-tuned T5-based question generation model as both the agent of RL and a stochastic parameterized policy π; in timestep t, the state s_t is defined as the generated tokens so far $(y_1, ..., y_{t-1})$; the action a_t is the next token to be generated (y_t) and the reward represents a score measuring how well the question is generated by the agent. We let the state transition be deterministic after an action has been chosen. Since we want to evaluate the generated question as a sequence but not a series of tokens, the reward can only be calculated after the complete question is generated. The overall task is defined to optimize the expectation of reward function $r : X \times Y \to R$:

$$\mathbb{E}_\pi(r) = \mathbb{E}_{x\sim D, y\sim\pi(\cdot|x)}[r(x, y)], \tag{5.2}$$

where D denotes the data distribution. To keep π from moving too far from old policy π_{old}, a penalty with expectation $\beta KL(\pi, \pi_{old})$ is added to the reward function:

$$R(x, y) = r(x, y) - \beta \log \frac{\pi(y|x)}{\pi_{old}(y|x)} \tag{5.3}$$

The reward for the generated question is named "answerable score", which means that the generated question should be a valid question that can be answered when only referring to the corresponding context C (the data input to the T5 encoder). We propose two versions of the answerable score based on the SpanBERT [60] regarding two different properties: confidence and correctness. SpanBERT is an extractive machine reading model that predicts the probability distribution of the start and end position (P_{start}, P_{end})of the answer in the context. The confidence-based answerable score R_{conf} is motivated by that if the question generation model is well-trained, then

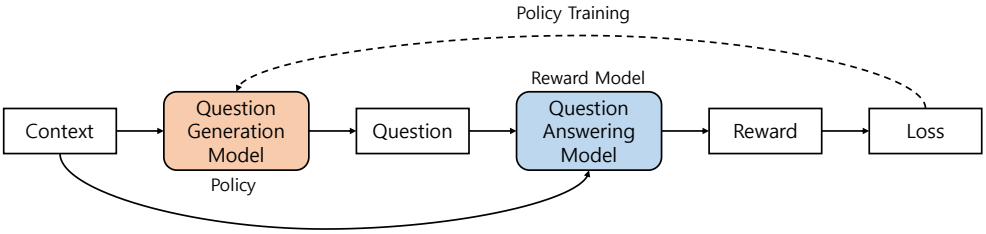

Figure 5.6: Illustration of sequence-level training of T5 model for question generation.

it should be aware of the pre-defined answer in the context and then generate a proper question. In the same way, if the question Q is well-generated, then the QA model also should be confident about the predicted answer span (confidence). Since we already know the desired answer, we can either define the correctness-based answerable score R_{corr} as how much the QA model is sure about the actual answer span (correctness):

$$R_{conf} = -log(1 - \max_{i \leq j \leq l} \sqrt{P_{start}(i|C,Q) \cdot P_{end}(j|C,Q)}) \tag{5.4}$$

$$R_{corr} = -log(1 - \sqrt{P_{start}(i_{answer}|C,Q) \cdot P_{end}(j_{answer}|C,Q)}), \tag{5.5}$$

where l is the length of the given context, i_{answer} and j_{answer} is the position index of start token and end token of answer span inside the context. The overall process of sequence-level fine-tuning with question-specific reward is shown in Figure 5.6.

5.4 Experiments

Following the experimental settings in Chapter 3, we evaluate our proposed method on SQuAD 1.1 dataset (split-1) with three automatic evaluation metrics: BLEU-4, Meteor, and Rouge$_L$. The proposed method (T5-small-qg-ppo) is compared to ASs2s [3], T5-small with only pre-training, T5-small with token-level fine-tuning (T5-small-qg). In addition, qualitative analysis is provided for comprehensive evaluation since the automatic evaluation metrics can only reflect a partial characterization of the models.

Table 5.1: Experimental results on BLEU-4, METEOR and ROUGE-L. **T5-small**: T5-small model with only pre-training. **T5-small-qg**: T5-small model with token-level fine-tuning. **T5-small-qg-ppo**: T5-small-qg + sequence-level training.

Model	BLEU-4	METEOR	ROUGE-L
ASs2s [3]	16.20	19.92	43.96
T5-small	2.45	10.66	12.92
T5-small-qg	19.12	25.39	45.21
T5-small-qg-ppo (confidence)	17.32	24.06	42.25
T5-small-qg-ppo (correctness)	17.45	24.32	42.31

5.4.1 Implementation Details

We implement our models in Pytorch 1.7.1 along with Hugging Face Transformers 4.5.1 and train the model with a single GTX 1080 Ti. The hyperparameters of our proposed method are listed as follows.

The proposed method is based on T5-small model, a down-sized version of the original T5 with about 60M parameters. T5-small use embedding size $d_{model} = 512$, output size of feed-forward layers $d_{ff} = 2,048$, 8-headed attention, and only 6 layers each in the encoder and decoder. For token-level fine-tuning of the T5-small, we use AdaFactor [115] optimizer with the batch size of 128 (using gradient accumulation technique implemented in Transformers library). We adopt the same learning rate scheduling technique used in [7]. The maximum sequence length for encoder and decoder input is 512 and 32, respectively. For sequence-level fine-tuning, we use the PPO2 version of PPO from [116] and 4 PPO epochs per mini-batch. The batch size is set as 32, and dropout is not used for policy training. The learning rate is 1.41e-05.

5.4.2 Quantitative Analysis

Following [3], we first use the evaluation package published by [79] to compare the performance of question generation models with respect to three evaluation metrics. Table 5.1 shows the evaluation results of each model. As expected, T5-small underperforms other methods. Even the T5-small is pre-trained with hundreds of gigabytes of language data, it can only get a general understanding of the language but not in every detail. T5-small model with only token-level fine-tuning (T5-small-qg) achieves the best performance on all of the metrics. However, both sequence-level fine-tuned T5-small models (T5-small-qg-ppo) rather show score drop in all three metrics. Since those three metrics are originally designed to measure the degree of some kind of "exact match" (e.g., n-gram precision and longest common subsequence), they are comparatively suitable to measure supervised training methods rather than unsupervised one. We can also explain that sequence-level training based on PPO changes the generated question as a whole in terms of its meaning rather than every exact token. We further compare the generated question from the various methods in a qualitative way.

5.4.3 Qualitative Analysis

We randomly sample examples from the test data and compare the generated questions from each method except T5-small. In Table 5.2, we find that generated questions by T5-small-qg-ppo methods are not very similar to the golden question, which results in the lower (automatic) evaluation results. However, the questions generated by T5-small-qg-ppo look easier to be answered by reward model SpanBERT since many of the generated part is directly copied from the single sentence inside the context input that has the most of the information related to the pre-defined answer.

The second sample in Table 5.3 shows a little bit different aspects of the questions generated by the proposed method. While all the generated questions are almost the same question, the one generated by the T5-small-qg-ppo (confidence) provides the most detailed information. Furthermore, the method tries to explore to capture an

Table 5.2: Comparison between different question generation methods given the same input context. The pre-defined answer is highlighted in bold text

Context Input	... zahm was active in the catholic summer school movement, which introduced catholic laity to contemporary intellectual issues. his book **evolution and dogma** (1896) defended certain aspects of evolutionary theory as true, and argued. more over, that even the great church teachers thomas aquinas and ...
Models	Generated Questions
Ground Truth	what book did john zahm write in 1896?
ASs2s [3]	what theory did darwin argue about aspects of evolutionary theory as true?
T5-small-qg	what was the name of john augustine zahm's 1896 book?
T5-small-qg-ppo (confidence)	what book defended certain aspects of evolutionary theory as true?
T5-small-qg-ppo (correctness)	what book defended certain aspects of evolutionary theory as true?

additional feature (new york) from other surrounding sentences, though it is wrong in this case. On the contrary, in this sample, T5-small-qg-ppo (correctness) generates the same question as the T5-small-qg, which aligns with the pre-defined answer. This can be explained as the correctness version of PPO training is more conservative than the one with confidence. We find some naming errors in the question generated by ASs2s, such as "victoria" should be beyoncé, and this is due to the fact that ASs2s gets sequence-level input while others get the whole context input.

Table 5.3: Comparison between different question generation methods given the same input context. The pre-defined answer is highlighted in bold text

Context Input	... on january 7, 2012, beyoncé gave birth to her first child, a daughter, blue ivy carter, at lenox hill hospital in new york. five months later, she performed for four nights at **revel atlantic city's ovation hall** to celebrate the resort's opening, her first performances since giving birth to blue ivy. ...
Models	Generated Questions
Ground Truth	her first appearance performing since giving birth was where?
ASs2s [3]	where did victoria perform for four nights at the resort's opening?
T5-small-qg	where did beyoncé perform for four nights?
T5-small-qg-ppo (confidence)	where did beyoncé perform for four nights to celebrate the opening of the new york resort?
T5-small-qg-ppo (correctness)	where did beyoncé perform for four nights?

5.5 Conclusion

In this chapter, we investigate the way of sequence-level training for conditional text generation. Specifically, we apply PPO algorithm, a sort of policy gradient method, to the question generation problem. The proposed methods are based on T5-small, one of the state-of-the-art seq2seq pre-trained language models. We define two types of reward functions for sequence-level training, each of which concentrates on the

confidence and correctness of the question generation models. We evaluate the proposed methods in both quantitative/qualitative ways. Experimental results show that the sequence-level training can help generate better questions given the appropriately defined reward.

Chapter 6

CONCLUSION

In this dissertation, we investigate various neural text generation algorithms concerning different training paradigms. We first introduce the general concept and the background knowledge related to neural network-based text generation algorithms. Then, we tackle the token-level training method for question generation task, which has suffered from a crucial issue that a significant proportion of the generated questions include words in the pre-defined question target (answer), resulting in the generation of unintended/unnatural questions. The proposed method, Answer-Separated Seq2Seq, encodes the passage and the target answer separately in an explicit manner. Furthermore, we propose another module called keyword-net which effectively extracts key information from the target answer. Experimental results show that our proposed method efficiently prevents the answer inclusion problem, resulting in the generation of better questions.

We find that the token-level training of autoregressive text generation models has several limitations, such as exposure bias and the rigorous training objective. We propose to adopt sequence-level training with a GAN-based approach to mitigate the issue. The proposed ConcreteGAN promotes the collaborative training of continuous-space methods and discrete-space methods. Experimental results on standard benchmark datasets show that the ConcreteGAN substantially outperforms state-of-the-art

text GANs with regard to quality, diversity, and global consistency.

Finally, we investigate the adoption of sequence-level training methods for the question generation problem. Based on a pre-trained language model, one of the biggest breakthroughs in NLP research, we apply an RL-based method instead of maximum-likelihood estimation. Considering the characteristic of the question generation task, we define two task-specific scores called "answerable score" as the reward for RL training. With the feedback from sequence-level scoring, the question generation model is demonstrated to have the ability to generate more informed questions.

Chapter 7

APPENDIX*

7.1 Generated Samples

We present samples generated by our proposed ConcreteGAN trained on COCO, EMNLP, and SNLI dataset in Table 7.1, Table 7.2 and Table 7.3 respectively.

Table 7.1: Randomly generated samples by ConcreteGAN trained on COCO dataset

1. red stoplights on a snow covered field .
2. a plate filled with pasta and a cake on a tray .
3. a cat stands on the hood of a car .
4. a black and white photo of a room with a ladder in it .
5. a jet plane sits on the runway while the sun rises or sets .
6. a car is decorated in the dirt by a beach .
7. i see into the sky beyond the light .

Table 7.2: Randomly generated samples by ConcreteGAN trained on EMNLP dataset

1. however , the number of residents affected by the emergency care will be released until april , the end of the worst record of the death .
2. if they have lost their families , they will be able to get back home and hopefully we ' ll come together with them .
3. there are some a little bit of interest in the coming months , it ' s unclear , " he said .
4. 18 - year - old actress said : ' i try to fight my life in the end of the war , but i ' ve never done that .
5. i was on the phone and i learned that i wanted to talk about you because you don ' t have any of it .
6. he cannot be registered with an independent parliamentary party , but it does not have to be protected by even if they receive an act of support .
7. the books , they did not deserve to be able to do something , but i don ' t think the worst .

Table 7.3: Randomly generated samples by ConcreteGAN trained on SNLI dataset

1. a woman wearing traditional clothing is posing for a picture .
2. the man is playing an instrument for a crowd of people .
3. a man is giving a speech to another man at the street .
4. soccer players are starting a struggle in the middle of the olympics .
5. two dogs sleep in a cage .
6. the man in a suit is asleep at a table .
7. two people are painting pictures of a mountain .

7.2 Comparison of ARAE and ARAE*

We compare our implementation of ARAE* and the original ARAE model with SNLI dataset, since the author of the ARAE published the pretrained model. In terms of FD score, ARAE achieves 0.011 which is the same as the score of ARAE*. We further compare the BLEU and the B-BLEU score of them. Table 7.4 shows that our implementation of ARAE* outperforms the original ARAE with respect to sentence quality and diversity.

Table 7.4: Comparison of the quality and diversity of sentences generated by ARAE and ARAE* with BLEU and B-BLEU scores on the Stanford Natural Language Inference dataset. A larger value indicates the better quality or diversity

Metrics	ARAE	ARAE*
BLEU-2	0.847	0.848
BLEU-3	0.620	0.639
BLEU-4	0.404	0.422
BLEU-5	0.263	0.272
B-BLEU-2	0.817	0.821
B-BLEU-3	0.627	0.635
B-BLEU-4	0.429	0.435
B-BLEU-5	0.282	0.288

7.3 Human Evaluation Criteria

The Human evaluation is based on grammatical correctness and meaningfulness and any text formatting problems (e.g., capitalization, punctuation, spelling errors, extra spaces between words and punctuations) are ignored. Workers are asked to score each sample based on the criteria shown in Table 7.5.

Table 7.5: Human evaluation criteria for 1-5 scoring

Score	Criterion
5 - Excellent	It's Grammatically correct and makes sense. For example: *"if England wins the World Cup next year, it will be the most significant result the sport has seen in more than a decade."*
4 - Good	It has some small grammatical errors and mostly make sense. For example: *"it is useful to have had a doctor who forced her to release him a couple of days before she was cleared."*
3 - Fair	It has major grammatical errors but the whole still conveys some meanings. For example: *"even then once again there's a sign of that stuff is going on the way to work on Christmas eve."*
2 - Poor	It has severe grammatical errors and the whole doesn't make sense, but some parts are still locally meaningful. For example: *"we go to work for the moment in life their eyes and, i have been a different race on to go."*
1 - Unacceptable	It is basically a random collection of words For example: *"i go com com com, i on on on play can go go."*

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the International Conference on Learning Representations*, 2015.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Annual Conference on Neural Information Processing Systems*, 2017.

[3] Yanghoon Kim, Hwanhee Lee, Joongbo Shin, and Kyomin Jung, "Improving neural question generation using answer separation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 6602–6609.

[4] Yanghoon Kim, Seungpil Won, Seunghyun Yoon, and Kyomin Jung, "Collaborative training of gans in continuous and discrete spaces for text generation," *IEEE Access*, 2020.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "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)*, 2019, pp. 4171–4186.

[6] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le, "Xlnet: Generalized autoregressive pretraining for language

understanding," *Advances in Neural Information Processing Systems*, vol. 32, pp. 5753–5763, 2019.

[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, pp. 1–67, 2020.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[10] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio, "Attention-based models for speech recognition," *Advances in neural information processing systems*, vol. 28, pp. 577–585, 2015.

[11] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[12] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli, "wav2vec: Unsupervised pre-training for speech recognition," *Proc. Interspeech 2019*, pp. 3465–3469, 2019.

[13] Yoon Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[14] Phil Blunsom, Edward Grefenstette, and Nal Kalchbrenner, "A convolutional neural network for modelling sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 52nd Annual Meeting of the Association for Computational . . . , 2014.

[15] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[16] Yanghoon Kim, Hwanhee Lee, and Kyomin Jung, "Attnconvnet at semeval-2018 task 1: Attention-based convolutional neural networks for multi-label emotion classification," in *Proceedings of The 12th International Workshop on Semantic Evaluation*, 2018, pp. 141–145.

[17] Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun, "Adversarially regularized autoencoders," in *International Conference on Machine Learning*, 2018, pp. 5902–5911.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

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

[20] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al., "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.

[21] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin, "Convolutional sequence to sequence learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1243–1252.

[22] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models.," in *AAAI*, 2016, vol. 16, pp. 3776–3784.

[23] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao, "Deep reinforcement learning for dialogue generation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1192–1202.

[24] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C Courville, and Yoshua Bengio, "A hierarchical latent variable encoder-decoder model for generating dialogues.," in *AAAI*, 2017, pp. 3295–3301.

[25] Lifeng Shang, Zhengdong Lu, and Hang Li, "Neural responding machine for short-text conversation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 1577–1586.

[26] Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al., "Towards a human-like open-domain chatbot," *arXiv preprint arXiv:2001.09977*, 2020.

[27] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky, "Adversarial learning for neural dialogue generation," in *Proceedings of*

the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 2157–2169.

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

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

[30] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li, "Incorporating copying mechanism in sequence-to-sequence learning," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1631–1640.

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

[32] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu, "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 11328–11339.

[33] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio, "Show, attend and tell: Neural

image caption generation with visual attention," in *International conference on machine learning*. PMLR, 2015, pp. 2048–2057.

[35] Andrej Karpathy and Li Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[36] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[37] Rico Sennrich, Barry Haddow, and Alexandra Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[38] Taku Kudo and John Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 66–71.

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

[40] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, pp. 9, 2019.

[42] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al., "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[43] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[44] Jeremy Howard and Sebastian Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 328–339.

[45] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[46] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher, "Non-autoregressive neural machine translation," *arXiv preprint arXiv:1711.02281*, 2017.

[47] Jason Lee, Elman Mansimov, and Kyunghyun Cho, "Deterministic non-autoregressive neural sequence modeling by iterative refinement," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 1173–1182.

[48] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014, pp. 103–111.

[49] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[50] Sam Wiseman and Alexander M Rush, "Sequence-to-sequence learning as beam-search optimization," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1296–1306.

[51] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.

[52] Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf, "Language generation with recurrent generative adversarial networks without pre-training," *arXiv preprint arXiv:1706.01399*, 2017.

[53] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[54] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[55] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[56] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun, "Adversarial ranking for language generation," in *Advances in Neural Information Processing Systems*, 2017, pp. 3155–3165.

[57] Sandeep Subramanian, Sai Rajeswar Mudumba, Alessandro Sordoni, Adam Trischler, Aaron C Courville, and Chris Pal, "Towards text generation with adversarially learned neural outlines," in *Advances in Neural Information Processing Systems*, 2018, pp. 7551–7563.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[59] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[60] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[61] Minh-Thang Luong, Hieu Pham, and Christopher D Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[62] Philipp Koehn and Rebecca Knowles, "Six challenges for neural machine translation," in *Proceedings of the First Workshop on Neural Machine Translation*, 2017, pp. 28–39.

[63] Michael Heilman and Noah A Smith, "Good question! statistical ranking for question generation," in *Human Language Technologies: The 2010 Annual Con-*

ference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2010, pp. 609–617.

[64] Nan Duan, Duyu Tang, Peng Chen, and Ming Zhou, "Question generation for question answering," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 866–874.

[65] Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou, "Question answering and question generation as dual tasks," *arXiv preprint arXiv:1706.02027*, 2017.

[66] Duyu Tang, Nan Duan, Zhao Yan, Zhirui Zhang, Yibo Sun, Shujie Liu, Yuanhua Lv, and Ming Zhou, "Learning to collaborate for question answering and asking," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, vol. 1, pp. 1564–1574.

[67] Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende, "Generating natural questions about an image," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, vol. 1, pp. 1802–1813.

[68] Xinya Du, Junru Shao, and Claire Cardie, "Learning to ask: Neural question generation for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, vol. 1, pp. 1342–1352.

[69] Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou, "Neural question generation from text: A preliminary study," in *National CCF Conference on Natural Language Processing and Chinese Computing*. Springer, 2017, pp. 662–671.

[70] Linfeng Song, Zhiguo Wang, Wael Hamza, Yue Zhang, and Daniel Gildea, "Leveraging context information for natural question generation," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018, vol. 2, pp. 569–574.

[71] Reinald Kim Amplayo, Seonjae Lim, and Seung-won Hwang, "Entity commonsense representation for neural abstractive summarization," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, vol. 1, pp. 697–707.

[72] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio, "Pointing the unknown words," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, vol. 1, pp. 140–149.

[73] Shuming Ma, Xu Sun, Wei Li, Sujian Li, Wenjie Li, and Xuancheng Ren, "Query and output: Generating words by querying distributed word representations for paraphrase generation," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, vol. 1, pp. 196–206.

[74] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang, "Squad: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2383–2392.

[75] Zhiguo Wang, Wael Hamza, and Radu Florian, "Bilateral multi-perspective matching for natural language sentences," in *Proceedings of the 26th Interna-*

tional Joint Conference on Artificial Intelligence. AAAI Press, 2017, pp. 4144–4150.

[76] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[77] Jeffrey Pennington, Richard Socher, and Christopher Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[78] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *International Conference for Learning Representations*, 2015.

[79] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco captions: Data collection and evaluation server," *arXiv preprint arXiv:1504.00325*, 2015.

[80] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," in *Proceedings of the International Conference on Learning Representations*, 2018.

[81] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[82] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut, "Text summarization techniques: a brief survey," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 10, 2017.

[83] Cyprien de Masson d'Autume, Shakir Mohamed, Mihaela Rosca, and Jack Rae, "Training language gans from scratch," in *Advances in Neural Information Processing Systems*, 2019, pp. 4300–4311.

[84] Sidi Lu, Lantao Yu, Siyuan Feng, Yaoming Zhu, and Weinan Zhang, "Cot: Cooperative training for generative modeling of discrete data," in *International Conference on Machine Learning*, 2019, pp. 4164–4172.

[85] Matt J Kusner and José Miguel Hernández-Lobato, "Gans for sequences of discrete elements with the gumbel-softmax distribution," *arXiv preprint arXiv:1611.04051*, 2016.

[86] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin, "Adversarial feature matching for text generation," in *International Conference on Machine Learning*, 2017, pp. 4006–4015.

[87] Md Akmal Haidar, Mehdi Rezagholizadeh, Alan Do Omri, and Ahmad Rashid, "Latent code and text-based generative adversarial networks for soft-text generation," 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)*, 2019, pp. 2248–2258.

[88] Alec Radford, Luke Metz, and Soumith Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *International Conference on Learning Representations*, 2016.

[89] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[90] Tero Karras, Samuli Laine, and Timo Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[91] Weili Nie, Nina Narodytska, and Ankit Patel, "Relgan: Relational generative adversarial networks for text generation," in *International conference on learning representations*, 2018.

[92] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio, "Maximum-likelihood augmented discrete generative adversarial networks," *arXiv preprint arXiv:1702.07983*, 2017.

[93] J Guo, S Lu, H Cai, W Zhang, Y Yu, and J Wang, "Long text generation via adversarial training with leaked information," in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. AAAI, 2018, pp. 5141–5148.

[94] Ronald J Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[95] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning, "A large annotated corpus for learning natural language inference," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 632–642.

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

[97] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu, "Texygen: A benchmarking platform for text generation models," in

The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 2018, pp. 1097–1100.

[98] Zhan Shi, Xinchi Chen, Xipeng Qiu, and Xuanjing Huang, "Toward diverse text generation with inverse reinforcement learning," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 4361–4367.

[99] Stanislau Semeniuta, Aliaksei Severyn, and Sylvain Gelly, "On accurate evaluation of gans for language generation," *arXiv preprint arXiv:1806.04936*, 2018.

[100] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al., "Universal sentence encoder for english," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 169–174.

[101] Laurens van der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[102] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, "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)*, 2018, pp. 2227–2237.

[103] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2019.

[104] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon, "Unified language model pre-training for natural language understanding and generation," *arXiv preprint arXiv:1905.03197*, 2019.

[105] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu, "Mass: Masked sequence to sequence pre-training for language generation," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5926–5936.

[106] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao, "Multi-task deep neural networks for natural language understanding," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 4487–4496.

[107] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7871–7880.

[108] Adina Williams, Nikita Nangia, and Samuel Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 1112–1122.

[109] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[110] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid, "Videobert: A joint model for video and language representation learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7464–7473.

[111] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988.

[112] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[113] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[114] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al., "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[115] Noam Shazeer and Mitchell Stern, "Adafactor: Adaptive learning rates with sublinear memory cost," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4596–4604.

[116] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov, "Openai baselines," `https://github.com/openai/baselines`, 2017.

초 록

자연어 처리 연구는 딥 뉴럴넷의 도입으로 인해 대대적인 발전을 거쳤다. 자연어 처리 연구의 일종인 자연어 생성은 기계가 내린 결정을 사람이 이해할 수 있도록 전달하는 기능이 있다, 그렇기에 사람을 모방하는 인공지능 시스템을 구축하는 데에 있어 필수 불가결한 요소이다. 일반적으로 뉴럴넷 기반의 텍스트 생성 태스크에서는 자동회귀 방법론들이 주로 사용되는데, 이는 사람의 언어 생성 과정과 유사한 양상을 띠기 때문이다. 본 학위 논문에서는 두 가지 뉴럴넷 기반의 자동회귀 텍스트 생성 모델 학습 기법에 대해 제안한다. 첫 번째 방법론에서는 토큰 레벨에서의 질문 생성 모델 학습 방법에 대해 소개한다. 논문에서 제안하는 답변 분리 시퀀스-투-시퀀스 모델은 기존에 존재하는 질문 생성 모델로 생성된 질문이 답변에 해당하는 내용을 포함하는 문제점을 효과적으로 해결한다. 주로 최대 우도 추정법을 통해 학습되는 자동회귀 방법론에는 노출 편향 등과 같은 문제점이 존재한다. 이러한 문제점을 해결하기 위해 논문에서는 텍스트의 연속 공간 표현과 이산 공간 표현 모두에 대해 상호보완적으로 학습하는 시퀀스 레벨의 적대 신경망 기반의 텍스트 생성 기법을 제안한다. 마지막으로 앞선 방법론들을 종합하여 시퀀스 레벨의 질문 생성기법을 제안하며, 이러한 과정에서 최신 자연어 처리 방법 중 하나인 사전 학습 언어 모델과 근위 정책 최적화 방법을 이용한다.

주요어: 딥 뉴럴넷, 텍스트 생성, 질문 생성

학번: 2014-22546

ACKNOWLEGEMENT

I would like to express my gratitude and appreciation for my supervisor Prof. Kyomin Jung whose guidance, support and encouragement has been invaluable throughout this study. I also wish to thank MILAB members for their collaborative effort during this research. To conclude, I cannot forget to thank my family and friends for all the unconditional support in this very intense academic year.