An Empirical Study on Learning and Improving the Search Objective for Unsupervised Paraphrasing

by

Weikai Steven Lu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Research in unsupervised text generation has been gaining attention over the years. One recent approach is local search towards a heuristically defined objective, which specifies language fluency, semantic meanings, and other task specific attributes. Search in the sentence space is realized by word-level edit operations including insertion, replacement, and deletion. However, such objective function is manually designed with multiple components. Although previous work has shown maximizing this objective yields good performance in terms of true measure of success (i.e. BLEU and iBLEU), the objective landscape is considered to be non-smooth with significant noises, posing challenge for optimization.

In this dissertation, we address the research problem of smoothing the noise in the heuristic search objective by learning to model the search dynamics. Then, the learned model is combined with the original objective function to guide the search in a bootstrapping fashion.

Experimental results show that the learned models combined with the original search objective can indeed provide a smoothing effect, improving the search performance by a small margin.

To my parents and my brother
For always giving me unconditional love and support.

The computer was born to solve problems that didn't exist before.

– Bill Gates.

Acknowledgements

First and formost, I would like to thank my supervisor Professor Lili Mou for his valuable guidance and feedback throughout my research. His insight and knowledge into the subject matter was invaluable in helping me formulating the research questions and methodology, as well as pushing me to sharpen my thinking and brought my work to a higher level. I would also like to pay my regards to my committee Dr. Alona Fyshe and Dr. Davood Rafiei for providing their valuable suggestion for my thesis.

I would like to thank everyone who supported me during my master program at the University of Alberta. I would like to thank Professor Martin Jagersand, whom I was fortunate to work with during my undergraduate study at the University of Alberta before entering the master program. He introduced me into research in computing science, and provided me with opportunities and horizons that I needed to choose the right direction in my research. I would like to thank all the friends I made during my two years of master program at University of Alberta who have shared with me priceless experiences and stories: Anup Anand Deshmukh, Chen Jiang, Chenyang Huang, Jiabin Fan. Special thanks to Mu He for being my best friend in life and always providing me with the warmest support.

I am deeply grateful to Wyatt Praharenka. Wyatt and I spent numerous late nights on campus combating school works. The support we have for each other helped us through innumerable down times, and we are now both graduating with our master's degrees.

Finally, I would like to express my utmost gratitude and appreciation for my family. Thanks to my mother Muyi, for all her selfless love and caring throughout the years. Thanks to my father, Jianjun, for his invaluable life experience and support. Thanks to my brother, Tony, for all the good memories we had together.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Natural language generation (NLG) has been a long standing task in the field of Natural Language Processing (NLP) over the years. Being able to understand, model and generate texts that are syntactically compliant, semantically meaningful, coherent with contexts, and free from rigid rules and artificial construction remains a challenge in NLP research.

A wide range of applications are dependent on text generation modules including dialogue systems, data-to-text generation, document summarization, sentence simplification, style transfer, and machine translation.

In this dissertation, we focus on conditional text generation, whose goal is to generate a piece of text based on a given piece of source text, while enforcing some task specific attributes. For example, the desired attributes for paraphrase generation task would be preserving semantic meaning from the input text, while using different wording; for machine translation, the generated text is required to deliver the same semantic meaning as the input text, while being syntactically coherent.

Due to the complexity of language generation tasks, traditional approaches rely on rules and templates. However, modern languages evolve over time through the pragmatic use by humans without conscious planning and pre-

meditation [28]. In other words, natural languages used by humans are fluid in terms of grammar and structures. For this reason, rules and templates are oftentimes restricted to narrow applications: for example, rules and templates used by a hotel booking chatbot are very different from that of an internet technical support service agent, leading to the need of sophisticated task-specific design for each application. Despite the high efficiency and controllability, rules and templates are never able to generate true natural language that are diverse, complex, while being able to deliver important information.

Recent advances in computational hardware and large scale machine learning models have revolutionized the paradigm of text generation. State-of-the-art language models powered by Artificial Neural Networks (ANN) have demonstrated their superior capability of modeling complex environmental states in various tasks. Language models powered by ANNs can now learn and represent complex natural languages as a black-box by learning from a large amount of training data. Such a data-driven approach to model languages eliminates the need of static rules and templates hand-crafted by humans, as the languages are modeled implicitly by the neural networks learning from true natural language data directly. Being able to model natural languages in such a free form has enabled numerous breakthroughs in NLP.

The most widely used approach for supervised text generation nowadays is the neural network-based sequence-to-sequence (Seq2Seq) model [42]. The role of Seq2seq model is to produce a mapping between sequential data. A Seq2Seq model first encodes the language input into a numerical representation using a neural network-based encoder. Such representation carries the semantic meaning and structural information of the input sentence, which is then used to generate output text by the decoder. With parallel corpora available, training of such a model is typically done by maximizing the probability of generating the ground-truth output given the input. Various modules are

proposed to parameterize this encoding-decoding process, including recurrent neural networks (RNN), and the more advanced Transformer [43] architecture with attention mechanisms.

However, parallel corpora are labor-intensive to acquire, and many applications do not come with a massive amount of parallel training data. For example, many rare languages or tasks are not well studied yet in the field of NLP, thus simply do not have large amount of annotated data for training supervised models. Another common scenario is when a model is to be used in a new domain, all training corpora dedicated for the old application is no longer usable, which is often the case when a company needs to quickly develop a minimum viable product for new applications. Being able to generate text without supervision by parallel corpora is very much in need in a lot of scenarios.

Unsupervised text generation has attracted increasing interest in the field of NLP over the years. One intuitive approach is the variational auto-encoder [4], which generates text by sampling from a latent space. However, low interpretability and controllability of the latent space pose challenges on controlling the text being generated. Consider the example of sentence simplification, where the goal is to shorten a sentence while preserving the original content. This goal would not be trivial to specify in the latent space, as it would not be apparent how to modify the latent representation would shorten the sentence length without compromising the content.

Another approach to unsupervised text generation is stochastic search towards a heuristically defined objective function. The objective function measures the quality of generated text in terms of general attributes including language fluency, expression fluency, and other task specific attributes such as length and expression diversity. Then the goal of the search algorithm is to find an output that maximizes this objective function. To accomplish

this, the discrete search algorithm takes small steps in the sentence space by making word-level editing, e.g., insertion, deletion, and replacement. Note that with search steps defined this way the dimension of search states grows exponentially with the vocabulary size. Hence, an exhaustive search is infeasible. Efficient heuristic search would be the more appropriate algorithm. The search algorithm can be plugged in with simple Hill-climbing or other non-greedy variants such as simulated annealing. This iterative local search process will explore the sentence space within the budgeted time steps, before settling with a locally optimal solution.

Stochastic search formulation of unsupervised text generation has demonstrated its flexibility and capability for a variety of tasks. Applications of the search-based framework in paraphrase generation [25], document summarization [40], and sentence simplification [22] have achieved state-of-the-art performance in respective tasks. Each of these models is equipped with both general and task-specific objective functions to guide the search towards appropriate output for the given task. Since local search algorithms such as hill-climbing (HC) and simulated annealing (SA) generate text by directly modifying the input text, search-based frameworks are especially suitable for tasks with significant overlap between input and output.

Despite the success of the search-based framework in a variety of tasks, there are some drawbacks to be tackled. In the aforementioned search-based framework, quality of the generated text is ensured by the heuristically defined objective function, which could provide general guidance on a population level, but might not be specifically instructive for each single sentence. More precisely, the objective functions are believed to correlate with the evaluation metrics on average for a large number of samples, but such correlation is weak when it comes to each individual instance. Hence, the search algorithms would have to face a very non-smooth objective landscape when searching for solu-

tion. Detail analysis is presented in the experiment section.

The non-smooth function landscape poses two main challenges for the search algorithm: guidance by the objective function may be inaccurate, meaning that the objective function which the search algorithm aims to maximize is skewed from the true measure of success; search algorithms would be more susceptible to local optimum in a non-smooth function landscape, leading to the search frequently getting trapped.

1.2 Thesis Statement and Contributions

In this dissertation, we address the research problem of smoothing the objective function for word-level local search, in order to improve search-based unsupervised paraphrase generation. More specifically, we claim that:

In the context of heuristic search-based text generation, learning of the search dynamic can help smoothing the objective function in a bootstrapping fashion, thus improving the performance of text generation in terms of BLEU and iBLEU evaluation metrics.

To attain our objective, we make use of deep neural networks to learn and model the search dynamic. Firstly, the search algorithm would search in the sentence space in order to generate search trajectory samples. Then, we train our deep neural network-based models using the collected search trajectory samples. Finally, the learned models will be used to guide the search algorithm together with the original objective function for the final generation of text. All of these are done in the unsupervised setting. Qualitative and quantitative experiments and analysis will be performed to support the above statement.

In this dissertation, we propose three approaches for learning the search dynamic of search-based text generation. In particular, we employ the Transformer architecture as the backbone for all three of our models. For the final inference, the learned model will be combined with the original objective func-

tion to guide the search.

In summary, the main contributions of this dissertation are as follows:

1. We reproduce a search-based text generation framework, UPSA [26], and validate its performance by evaluation metrics, serving as a testbed for our proposed models.

2. We collect, clean, and process a dataset of search trajectories to train our models.

3. We propose and implement three different novel learning-for-search text generation models. We evaluate and analyze the model quantitatively and qualitatively on the task of paraphrase generation, showing the effectiveness of our proposed approaches.

1.3 Thesis Organization

The thesis is organized as follows: in Chapter 2, we overview the current state of the art of natural language processing and natural language generation. In particular, we review how various kinds of deep neural networks are used to process, represent, and generate natural language. We further explain the general working mechanism of the Transformer backbone that we adopted. Search-based text generation framework is the focus of this dissertation, thus recent work in this area would also be covered in details in this chapter.

In Chapter 3 we detail our reproduction of the UPSA [26] model as our testbed, and propose three different models stemming from the same idea: two of them are BERT-based [6] regression models that aim to directly model the manually designed heuristic objective function, while the third model learns the search via a Transformer-based sequence-to-sequence model [42]. The learned model would be combined with the original objective function to guide

a second iteration of search to generate the final output. The motivation of this approach, and how it works with the UPSA model would be discussed in details. Moreover, we also describe how to collect and process a dataset needed to learn from the search dynamics of simulated annealing on the task of unsupervised paraphrase generation.

Experiment and analysis procedures and results are presented in Chapter 4. The main purpose of this chapter is to demonstrate the effectiveness of our proposed models. Following previous work in paraphrase generation, we adopt BLEU [33] and iBLEU [41] as measures of success. We also aim to gain insights on the underlying behavior of our proposed models, and verify our motivation of the approach. Specifically, we are interested in how the score predicted by our models correlate with true measure of success, compared with the original objective function; we would also look into how the acceptance ratio in simulated annealing is changed by our models; and finally, we want to investigate whether our proposed models indeed help the search algorithm avoid getting stuck at local optimum.

Finally, we conclude this thesis in Chapter 5 with a summarization of our finding and contribution. We also discuss our thoughts and understanding in retrospect, hoping to pinpoint future direction for search-based text generation.

Chapter 2

Background and Related work

2.1 Natural Language Generation

Natural language generation (NLG) has been a long-standing task in the field of natural language processing (NLP). In general, an NLG system aims to generate natural languages that are diverse and able to convey information of interest. NLG and natural language understanding (NLU) are the two important subfields of NLP research. Complementary to NLG, NLU modules are responsible for the understanding of natural language input and representing the semantic and logical relationship with the context. A large number of applications are achieved by the combination of NLG and NLU, yielding NLP systems that can take natural language as input from the user, then process the command or query with a database, and finally respond to the user in natural language as in Figure 2.1. Although usually taking up different roles in an NLP pipelines, NLU and NLG are by no means orthogonal: in order to generate natural language text, NLG systems need to develop an understanding of natural language. During the text generation process, the NLG system needs to maintain a representation of the semantic meaning or message to be delivered, then generate a text based on that representation. Hence, NLU can be considered as an important building blocks for NLG systems.

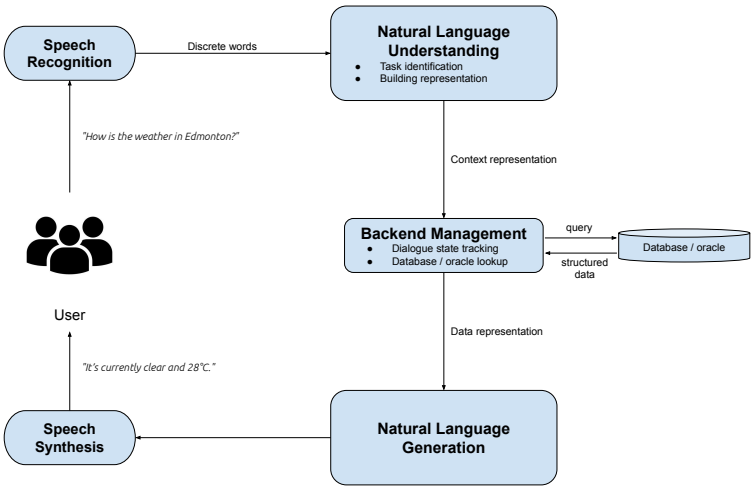


Figure 2.1: An example dialogue system pipeline.

2.1.1 Text Generation Applications

Various applications have been realized by NLG models. Perhaps the most sought after application of NLG is machine translation (MT) [2][27][45], whose goal is to generate text in the target language domain with same meaning as the input from the source language domain. Machine translation is not a trivial task as word-to-word substitution does not solve the problem due to the different grammar and expression style among various languages. To accomplish the task of automatic translation the system needs to understand the syntax and semantic of both the source language and target language, while developing a mapping between the two languages. Another commercially successful applications of NLG in recent times is data-to-text generation [44][35]. Generating weather report in natural language [8] is one of the earliest application of data-to-text generation, whilst nowadays numerous applications are made to interpret and summarize financial and business data [23][12]. Another application with rising popularity is dialogue systems, whose goal is to converse with human in natural language, and be able to answer the queries

or execute the commands according to the interaction history with the user. In this case, the NLG component would be crucial to process the retrieved data and able to deliver the gist to the user via data-to-text generation. For example, if a user requests "Summarize the news for me today.", the NLU component would need to understand the information to look up, and deliver the retrieved data to the NLG component, before the NLG component can perform the summarization task on the news. Oftentimes, the generated natural language response might require further polishing before sending out to user. For instance, the text generated by the aforementioned pipelines might require another model to paraphrase it to improve fluency and correct the grammar, or some sensitive words need to be filtered out.

Challenges of both NLU and NLG are rooted in the complex nature of human languages: compared with programming or scripted languages that come with rigid structures and unambiguous instruction, human languages are more likely to be ambiguous, diverse, context dependent, and even oftentimes syntactically erroneous. For this reason, natural languages with appropriate syntax and semantic patterns are in general difficult to specify by rules and templates. More specifically, the challenge of NLU is to produce a consistent representation for the same semantic meaning in various forms, while the challenge of NLG is to stay consistent throughout the process of emitting words without degeneration. Another major challenge for NLG is the vocabulary size. Take English language as an example. Typically, the vocabulary size is over 30,000. Considering the number of combinatorial sentences grows exponentially with the length by a factor of the vocabulary size, such high dimensionality renders exact solutions intractable even if the true measure of success is known.

2.1.2 Task Formulation

The general task of generating text can be formulated as follows: Given the source language domain $\mathcal{X}$, the target language domain $\mathcal{Y}$, and a task specific function $f(X, Y)$ indicating the quality of the generated text $Y \in \mathcal{Y}$ given the input $X \in \mathcal{X}$, the goal of text generation would be to find the best text Y^* that maximizes the task specific objective function $f(X, Y)$. For example, in machine translation $f(X, Y)$ would evaluate the combination of semantic similarity between the source text and translated text, fluency of the generated text, as well as how appropriate the tone is; for automatic document summarization, $f(X, Y)$ would be based on the preservation of the original meaning and some measure of brevity. However, such $f(X, Y)$ may not be easily defined due to the difficulty of evaluating complicated natural languages. Ideally this should be done by human evaluation. Oftentimes we might need to settle for an approximate evaluation metric that is largely representative of the conceptual idea, and feasible for automatic evaluation.

2.1.3 Modelling Text Generation

Traditionally, text generation is accomplished by rules and templates. For relatively narrow applications, filling the blank in a fixed template would be viable. For example, weather report is one of the more narrow domains where everyday almost the same data structure is fed to the text generation system to generate the weather report, meaning that it would suffice to have one or a few templates with the same blanks to be filled in by the fixed data structure (e.g., temperature and humidity). To handle slightly varying data structures, the filling-the-blank system can be embedded by scripting languages, in which one can specify rules that constitute the templates using conditional branching, or logical loops to handle slightly varying data structure. However, as seen such system is manually constructed by humans; thus, expressions are

relatively restrictive because the system would not be able to generate text that was not directly scripted by humans. Text generated in such a way is by no means diverse, and is considered not to be a form of natural language. As seen, although using templates and rules might work well in some narrow applications, such system would obviously fail when facing more complicated tasks that cannot be scripted for every instances, not to mention even if the scripts are achievable by intensive labor, they are impossible to generalized to different domains.

In recent years, the strong function approximation capability of deep neural networks (DNN) [9] has revolutionized the field of NLP. DNNs are complex composition of linear functions with non-linear composite connections that can theoretically approximate any function given enough memory and computational capacity [16]. Multi-layer perceptron is a common class of DNN architecture consisting of multiple layers of linear neurons that are parameterized by weights and bias, with non-linear synapses connecting layers to layers in one direction. Typically DNNs operate as feed-forward connectionist networks that take the input and let the numerical data flow through layers by layers, producing representations from each layer. Then the representation can be used by downstream decoders to generate text. Such design is originally inspired by how synaptic system in human brains transmit signals.

One key advantage of DNNs over traditional rules and templates is the elimination of manual feature engineering. DNNs are able to learn a representation of content automatically from data and use it to make prediction. Tremendous successes have shown the capability of DNNs learning meaningful features and using them to model noisy, uncertain, and high-dimensional environments.

The power of the DNNs comes at the price of needing huge amount of data for training. Typically DNNs are trained by trial and error. At each training

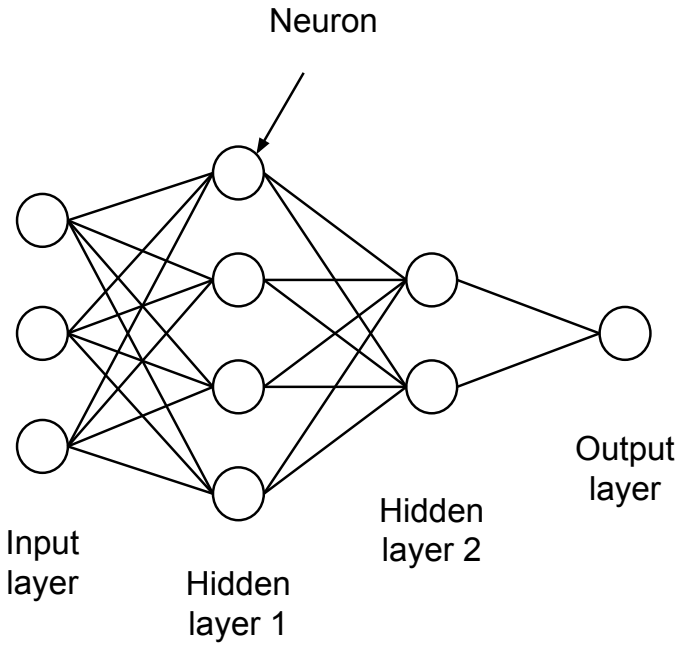


Figure 2.2: An example deep neural network.

step, the DNN makes a prediction, which is evaluated by a differentiable loss function to indicate how good the prediction is. Then, automatic learning of DNNs is accomplished by backpropagation (BP), which is the process of modifying the weight and bias parameters in all neurons in the direction of negative gradient with respect to the loss function by a small step. The intuition of modifying parameters in such a way is that weights moved in the direction of negative gradient lead to steepest descent of the loss function locally. By making small gradual local steps for a great many iterations, the loss function should be somehow minimized.

Learning and predicting the probability $P(w_1, \ldots, w_l)$ of a sequence $w_1, \ldots, w_l$ of length l are the backbones of various NLP applications, including speech recognition, part-of-speech tagging, and semantic parsing. Since text is made of discrete word tokens, such probability is usually factorized into $P(w_1, \ldots, w_l) = P(w_1) \cdot P(w_2|w_1) \cdot \cdots \cdot P(w_l|w_1, \ldots, w_{l-1})$. In other words, the probability of a given sentence is computed by the product of probability of all words

given all preceding words. Before the era of DNNs, lack of computational power calls for the need of a simplified approximation. Uni-gram and n-gram models are designed for this exact purpose, which essentially approximate $P(w_1, \ldots, w_l) = P(w_1) \cdot P(w_2|w_1) \cdot \cdots \cdot P(w_l|w_1, \ldots, w_{l-1})$ with truncated probabilities $P_{\text{uni-gram}}(w_1, \ldots, w_l) = P(w_1) \cdot P(w_2) \cdot \cdots \cdot P(w_l)$ and $P_{n\text{-gram}}(w_1, \ldots, w_l) = \prod_{i=1}^{l} P(w_i|w_{i-(n-1)}, \ldots, w_{i-1})$, respectively. Modern language models parameterized by DNNs are more expressive, and thus can fully model $P(w_1, \ldots, w_l) = P(w_1) \cdot P(w_2|w_1) \cdot \cdots \cdot P(w_l|w_1, \ldots, w_{l-1})$. In order to convert word tokens into numerical representation for DNNs, words are represented by word embeddings, which map discrete word tokens to continuous real-valued vectors while encoding semantic meaning of words in such a way that words with more similar meaning are mapped to vectors with a smaller distance, and vice versa. Representing words in such way provides more meaning than simply using indexes. Moreover, modelling a vocabulary of words in continuous space theoretically and practically alleviates the curse of dimensionality. Plenty of pre-trained embeddings are readily available nowadays, such as Word2vec [31] and GloVe [34], setting up the foundation for various applications.

2.1.4 Sequential Models

Recurrent neural network (RNN) is a family of DNN models that connects computational layers in a cyclic way over temporal steps to process varying-length sequences sequentially and finally outputs a representation for downstream tasks. As for vanilla RNNs, the outputs of the neurons at time step t are used as input for the neurons in time step $t + 1$. In such a way, each time step the representation is being processed using the same set of weights. Recurrent computation can be conceptually unrolled to reflect the feed-forward pass over time as in Figure 2.3. Theoretically, vanilla RNNs should have the power

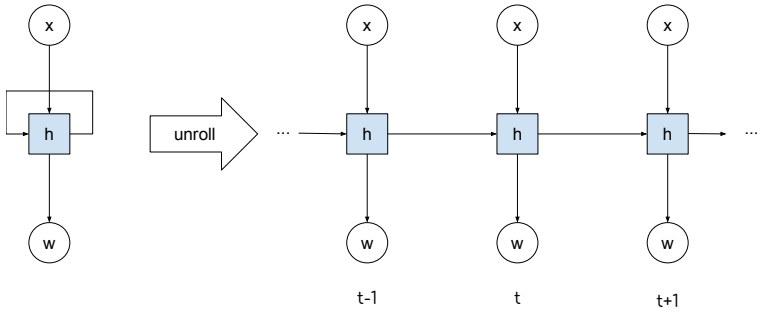


Figure 2.3: Recurrent neural network and unrolled perspective.

of modelling long-term dependencies. However, in practice vanilla RNNs face various problems, including gradient exploding and vanishing during training time [14][17]. Long-short term memory (LSTM) [15] recurrent neural networks alleviate these issues by regulating dependencies using a gating mechanism.

As seen, vanilla RNN is one-directional, meaning that when processing word w_t, only $w_1, \ldots, w_{t-1}$ are taken into consideration. Bidirectional RNNs is a simple direct improvement over vanilla RNNs. Instead of only having one hidden state accumulating information from the beginning of a sequence, bidirectional RNNs use another hidden state that accumulates information from the end of the sequence, providing more context when dealing with each single word in the sequence.

The sequence-to-sequence (Seq2seq) [42] model is a popular text generation model for various tasks, which essentially maps an arbitrary length text sequence to another arbitrary length sequence by compressing the former into a latent representation before generating output conditional on that. The Seq2seq model is originally developed for the task of machine translation due to its flexibility with different source and target language domains, but its simplistic and flexible design has made its way to numerous applications, such as part-of-speech (POS) tagging. Seq2seq models are typically constituted by an encoder, which can be parameterized by RNNs, LSTMs, or other sequential models, and a decoder that generates output sequentially. The encoder takes

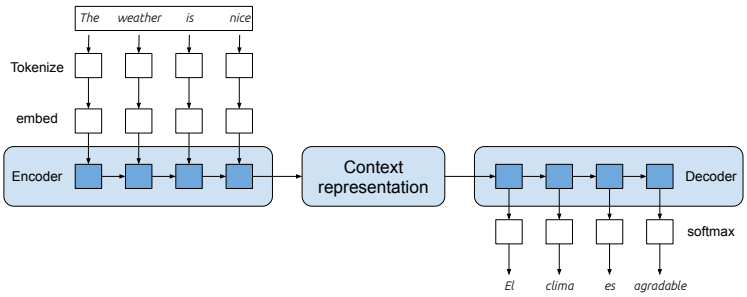


Figure 2.4: An example sequence-to-sequence (Seq2seq) model for English-to-Spanish machine translation.

an input sentence, and outputs a latent representation that carries the information from the input sentence. The decoder then takes the latent representation as input and generate text auto-regressively from the first word until a special stopping token is emitted, to deliver the content specified by the latent representation. At each time step, a probability distribution over the entire vocabulary conditional on the already generated tokens $P(w_t|w_1, \ldots, w_{t-1})$ is given by the softmax function in the decoder, indicating the probability of w_t following a sequence of $w_1, \ldots, w_{t-1}$, given the input sentence.

2.1.5 Attention Mechanism

Despite the effectiveness of the gating mechanism in LSTMs, RNN-based Seq2seq models still suffer from two major issues: RNNs are not parallelizable since the input for a time step strictly depends on the previous time steps, thus computation in all time steps must be done one-by-one; although the gating mechanism in LSTMs can alleviate the gradient vanishing problem, the gradient exploding problem implies the difficulty of feeding hidden state linearly forward over time steps (e.g., if w_{t+n} is heavily dependent on w_t, then the hidden state needs to carry the relevant information from time step t all the way through time steps t to $t+n$). The attention mechanism [2], inspired by cognitive science, is designed to tackle these issues. Instead of using the hidden state from strictly the previous time step (forward or backward), an

attention mechanism also makes use of the input text directly weighted by their relevance to the output. The attention weights given by an alignment model parameterized by DNN is a probability distribution indicating the "correlation" between all the words in input sentence and the specific word being processed.

Transformers [43] are a family of sequential model that uses attention only to process text and has been trending since it was first introduced. Similar to the aforementioned RNN-based Seq2seq model, Transformer-based Seq2seq models also follow the encoder-decoder architecture. Other than that, Transformers are very different from RNNs in terms of architecture and have several advantages. The core idea of Transformers is to use attention alone without recurrent structure to process sequential data to allow the model to process time dependencies without linear recurrent passage. Besides, the attention mechanism enables parallelization of computation, and thus improving training efficiency. The attention mechanism used by the original Transformer [43] is multi-head self-attention, meaning that when processing the input the model attends to the input itself, and having multiple attention heads functioning independently.

Various large-scale Transformer-based models have demonstrated their capability in a wide variety of tasks. Bidirectional Encoder Representation from Transformer (BERT) [6] is a set of models for NLP tasks developed by Google. The original English-language BERT models are pre-trained on the Bookcorpus [50][1] and the English Wikipedia dataset[2], consisting of 800M and 2.5B samples, respectively. BERT achieved state-of-the-art performance on a variety of NLU benchmarks, including General Language Understanding

[1]Bookcorpus dataset: `https://github.com/soskek/bookcorpus`

[2]English Wikipedia dataset: `https://en.wikipedia.org/wiki/Wikipedia:Database_download`

Evaluation (GLUE)[3], Standard Question Answering Dataset (SQuAD)[4], and Situations with Adversarial Generations (SWAG) [47], showing the superior capability of BERT in language understanding. Representations learned by BERT while training towards general purpose NLU tasks are now used as embeddings for various downstream tasks. Evidence [38] shows that pre-trained BERT models often performs well with only minimal fine-tuning on relevant NLU tasks. Generating text from BERT is not trivial, as BERT is inherently bidirectional, rendering auto-regressive word generation infeasible. However, there are various ways to decode text from BERT, such as starting with a sentence filled with empty blanks, then using BERT to fill all the blanks in a specified order [13]. Generative Pre-trained Transformer (GPT) [36] and its direct scaled up successor GPT-2 [37] are pre-trained Transformers geared more towards text generation. Different from BERT, GPT models have only decoder, but are also equipped with multi-head self-attention.

2.2 Edit-Based Text Generation

Different from the MT-based Seq2seq model that emits words one after another, edit-based text generation models refer to the family of models that learn to generate text by modifying the input text. Solving NLG tasks by predicting edit operation sequence has several advantage over auto-regressive text generators: many monolingual text generation tasks require generating text that have big overlap with the input. Take sentence simplification as an example: the easy way to simplify a sentence is to remove non-essential utterances while keeping all important key words, then fill in some new words to make the simplified sentence fluent and grammatically coherent. Using a MT-based Seq2seq model for this purpose would require the model to implicitly learn all

[3]GLEU:`https://gluebenchmark.com/`
[4]SQuAD: `https://rajpurkar.github.io/SQuAD-explorer/`

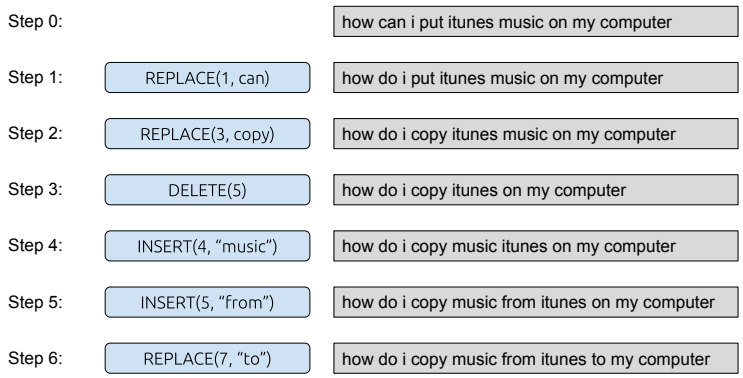


Figure 2.5: Example of paraphrase generation by edit.

these edit actions, while still be able to recover the essential keywords via the context representation. Indeed, experiment results show that Seq2seq models in this case are prone to generate the exact same output as the input most of the time, indicating that the minimal differences between inputs and outputs are oftentimes not captured by the Seq2seq models [49].

Learning the edit operations in the supervised setting requires first finding the edit operation sequence given parallel sentence pairs. This can be done by dynamic programming (DP) by first creating a table to store the results of sub-problems(i.e. partially edited text), then the minimum edit operation sequence can be obtained by tracking the table. Each valid edit label consists of a sequence of edit operation that if executed will convert the input sentence X to the target sentence Y. Commonly used word-level edit operations are insertion, deletion, and replacement. However, various operation formulations can be used to suit the need of various tasks. Edit-based approach allows the model to theoretically generate any sentence while also be able to preserve important keywords from the input sentence relatively easy by choosing not to modify those tokens. Moreover, such design also comes with more controllability and interpretability than MT-based Seq2seq models, as the generated operation sequence can reflect which part of the input needs to be edited instead of generating everything from scratch as a black-box.

2.2.1 Supervised Learning

In this section, we overview the current state of the art of edit-based text generation in supervised setting.

[1] aligns the input and output sentence to the best extent, before using heuristics to find the edit sequence label. The word-level edit operations they adopt are MOVE, DELETE, and REPLACE. Since heuristics are used instead of dynamic programming, annotations themselves are not exactly optimal, yielding 92% accuracy and an average of 0.7 F1 score compared with human expert annotations. Finally, they train a bi-directional LSTM-based RNN model to learn from the automatic annotated edit labels. Experiment results on sentence simplification task show that the RNN model does not learn the edit operations very well, but is still able to generate good quality sentence in terms of human evaluation.

EditNTS [7] adopts the neural programmer-interpreter model for sequential text edit operation prediction. The programmer module predicts the edit operation by jointly considering the partially generated sentence, edit trajectory, and contexts, while the interpreter is in charge of the realization of edit operations. Different from the RNN model [1], edit annotations in EditNTS are labeled by dynamic programming, which theoretically should yield 100% accuracy, superior to the heuristic based approach. For the task of sentence simplification, the programmer predicts one operation from ADD, DELETE, and KEEP for each word in the sentence. EditNTS model is trained to predict the correct edit label for each word by maximum likelihood training. The maximum likelihood training loss function can be modified to counter the class imbalance among ADD, DELETE, and KEEP operations by assigning different weight to each edit label in the loss function. This is also one way to inject human expert bias of preference among edit operations (i.e. to edit

more or to keep more words from the input sentence). Experiment results on sentence simplification task with WikiLarge, WikiSmall [48], and Newsela dataset [5] show that on all three dataset EditNTS consistently outperforms all other simplification models that does not use external knowledge base by the time of publication. More specifically, performance of EditNTS is consistently higher than that of the vanilla MT-based RNN model [1], demonstrating the superiority of edit-based text generation in sentence simplification task. We attribute the higher performance of EditNTS to two factors: quality of edit labels used by EditNTS are of higher quality since dynamic programming are used to create the labels; the programmer-interpreter model is more suitable than simple RNNs for learning edit labels.

Levenshtein Transformer [11] is an interesting mixture of edit-based model and auto-regressive model. The editing process is done by first processing input by a Transformer model, then iteratively delete, insert placeholder, and finally fill in the placeholder by word. The deletion, placeholder insertion, and fill-in-blank steps are each done by their own models at each step, but share the same Transformer backbone. Such iteration loops until the deletion and insertion policies stabilize or fixed budget is reached. In the experiment, Levenshtein Transformer achieves high performance and efficiency in machine translation, text summarization, and automatic post-editing.

LaserTagger [29] proposed an edit-based text generation model using pre-built phrase vocabulary. This model comes with a very simplistic three step design: encode input, tag edit operations, and realize the edits. The highlight of this work is the phrase vocabulary they adopt for phrase insertion. The phrases in the vocabulary are built from pair-wise alignment of all parallel pairs in a dataset, then the mismatched n-grams between input and ground-truth output after alignment are stored in the phrase vocabulary. The phrase

[5]Newsela dataset: `https://newsela.com/data/`

vocabulary has a much smaller size than a word vocabulary, and thus making the model eaiser to train. Using such a restricted vocabulary, LaserTagger yields similar performance compared with the BERT-based Seq2seq model when large number of training examples are available, but significantly outperforms the Seq2seq model when training examples are scarce (≤ 1000).

2.2.2 Unsupervised Text Generation

As seen, edit-based text generation models have several advantage over MT-based models in supervised learning setting. However, lack of parallel corpora in certain tasks call for unsupervised edit-based models.

The variational auto-encoder (VAE) [20] is a popular generative model for unsupervised setting. [4] proposes to generate text using VAE by sampling from the latent sentence space. Different from auto-encoders (AE) [3], VAE compresses the input (i.e. a sentence) into a mean μ and standard deviation σ that parameterize a multi-variate distribution, which is then used to sample the latent representation z. Then the LSTM-based decoder would generate text conditional on the latent representation. The training objective is to minimize the evidence lower bound, which measures the distance between the estimated posterior and the real posterior. Experiment results show that VAE is a competitive approach for imputing missing words and text classification. However, due to the black-box nature of the latent space, VAE would not be able to generate text with specific attributes.

Generating text by Metropolis-Hastings (MH) sampling in word space [30] is the cornerstone that leads to search-based NLG which this dissertation focuses on. Metropolis-Hastings algorithm is a Markov Chain Monte Carlo (MCMC) sampling algorithm that generates samples by iteratively jumping from state to state within a Markov Chain governed by a discrete probability distribution. Here the Markov Chain is defined to be the space of all sentence

$\mathcal{X}$, and each state is a sentence $x \in \mathcal{X}$. To explore the sentence space by sampling, the MH algorithm involves two steps: starting from any state x, first a candidate transition to x' is proposed using one of the word-level edit operations from insertion, replacement, and deletion; Then, the probability of accepting this proposed transition x' is determined by

$$A(x'|x) = \min\{1, A^*(x'|x)\} \tag{2.1}$$

$$\text{where} \quad A^*(x'|x) = \frac{\pi(x')g(x|x')}{\pi(x)g(x'|x)} \tag{2.2}$$

where $g(x'|x)$ is the *proposal function* that suggests a tentative transition x' from x, and $\pi(x'|x)$ is the *stationary distribution* that defines a probability distribution over the sentence space. As seen $A(x'|x)$ is rectified to the range of $[0, 1]$ to output a probability of accepting the transition to x'. In other words, the probability of not accepting transition to x' and remaining at x is $1 - A(x'|x)$. The motivation of MH algorithm is that sampling directly from $\pi(\cdot)$ is difficult as it involves computing $\pi(x)$ for all x to get a normalized measure. In the MH algorithm, only state at the sampling time x_t and the proposed neighbour x'_t require inference with $\pi(\cdot)$, thus bypassing the computational complexity of a global inference. For the task of text generation, $\pi(\cdot)$ is essentially a non-learnable heuristic function that takes as input a specific state x (the original input sentence x_0 is also taken as input implicitly), then evaluates a score indicating the quality of the state x with respect to the task. The idea of using MH sampling is to generate a sequence of states that follows the distribution $\pi(\cdot)$, such that sentences with relatively higher probability (i.e. higher quality sentence) are likely to be sampled more, while those with relatively low probability (i.e. lower quality sentence) are less likely to be sampled. In other words, MH sampling would be able to continually generate different sentence states by jumping around in the sentence space.

Model in [30] does not require any supervision since the distribution π

is manually designed by human to reflect the requirement for the text to be generated x^*. Consider the example of keyword-to-text generation. The distribution $\pi\cdot$ can be set to

$$\pi(\cdot) \propto p_{\text{LM}}(\cdot) \cdot s_{\text{keyword}}(\cdot) \tag{2.3}$$

where $p_{\text{LM}}(\cdot)$ is the probability given by a language model, indicating the fluency and grammatical coherence of a particular sampled state. More specifically, the language model for evaluating $p_{\text{LM}}(\cdot)$ is trained from non-parallel corpora in a unsupervised fashion; s_{keyword} is the hard keywords constraint taking value of 1 or 0, indicating if all required keywords are present in the state. This ensures the MH sampler never visit any state that does not strictly have all the required keywords present in the sentence.

Furthermore, for the task of paraphrase generation, the MH algorithm would sample from

$$\pi(\cdot) \propto \pi(\cdot) \propto p_{\text{LM}}(\cdot) \cdot s_{\text{match}}(\cdot|x_0) \tag{2.4}$$

where $p_{\text{LM}}(\cdot)$ is language model probability indicating the fluency of generated text, similar to Equation 2.3; $s_{\text{match}}(\cdot|x_0)$ is a matching score function indicating how similar are the semantic meaning between the generate sentence and original sentence to be paraphrase, which could be implemented by cosine similarity of pre-trained embeddings, or skip-thoughts sentence similarity [21]. Note that although manually designed heuristics are used, there are no explicit rules or templates being imposed. Components of $\pi(\cdot)$ (except for hard constraint) are all parameterized by pre-trained DNNs or embeddings, and thus able to recognize complicated semantic structures.

Experiment result in [30] shows that the MH sampler yields promising performance in keywords-to-sentence generation, unsupervised paraphrase generation, and unsupervised error correction, showing the generality of their model.

Hence, it can be concluded that it is feasible to manually design a task specific distribution $\pi(\cdot)$, then generate text via sequential sampling from such distribution.

$$x_1, x_2, \ldots, x_T \sim \pi(\cdot|x_0) \tag{2.5}$$

MH sampling is in a sense wasteful as it samples both high quality and low quality sentences. Although sentences of higher quality are more likely to be sampled, there are still quite some steps wasted on low quality region in the sentence space as in Equation 2.5. In other words, it would make more sense to use a sampling mechanism that focus on finding a maximum score sentence as in Equation 2.6.

$$x^* = \underset{x}{\text{argmax}}\pi(x|x_0) \tag{2.6}$$

In the next section, we overview search-based sampling for this exact purpose.

2.3 Search-Based Text Generation

In this dissertation, we refer search-based text generation models as word-level edit-based models that generate text by searching towards an objective function. Similar to [30], a manually specified objective function is used to indicate the quality of an arbitrary sentence for a given task. However, instead of sampling from a distribution, search-based model aims to optimize the objective function by searching towards its optimum using discrete word-level edit operations.

[40] tackles the task of document summarization by hill-climbing(HC) search, which ensures every accepted search step leads to a higher scored state. Summarization in this model is realized by word extraction, meaning that the model generates summary by directly using words from the original document. A good summary should be fluent, semantically similar to the original document, but has a shorter length. To specify these requirement, the objective

function designed for the summarization task is

$$f(\boldsymbol{y};\boldsymbol{x},s) = f_{\overrightarrow{\text{LM}}}(\boldsymbol{y}) \cdot f_{\text{SIM}}(\boldsymbol{y};\boldsymbol{x})^{\gamma} \cdot f_{\text{LEN}}(\boldsymbol{y};s) \tag{2.7}$$

where s is the desired length of summary; $f_{\overrightarrow{\text{LM}}}(\boldsymbol{y})$, $f_{\text{SIM}}(\boldsymbol{y};\boldsymbol{x})$, and $f_{\text{LEN}}(\boldsymbol{y};s)$ are components enforcing different requirements for generated text; and γ is a hyperparameter adjusting the relative weights of these components. $f_{\overrightarrow{\text{LM}}}(\boldsymbol{y})$ is the perplexity of sentence $\boldsymbol{y}$ given by a forward-backward language model, indicating the fluency of sentence $\boldsymbol{y}$; $f_{\text{SIM}}(\boldsymbol{y};\boldsymbol{x})$ measures the similarity between the original document and sentence $\boldsymbol{y}$, which is realized by computing the cosine distance of the sent2vec [32] embeddings of the two; and $f_{\text{LEN}}(\boldsymbol{y};s)$ is the hard constraint of length that outputs 1 if length sentence $\boldsymbol{y}$ is equal to s and 0 otherwise. The search space is defined to be a vector of Boolean variables $(a_1,\ldots,a_n)$ indicating if each specific word in the original document is extracted for the summary. To find a satisfactory summary in this search space, the model would iteratively sample one word at a time from the original document to swap a random word from the summary. The HC algorithm would only accept the new summary only if it is evaluated to have a higher score $f(\boldsymbol{y};\boldsymbol{x},s)$ than the current summary. This ensures that as the sampling process goes on, the quality of summary can only get better or stay the same. Experiment results on Gigaword and DUC2004 dataset show that such approach achieves a new stat-of-the-art performance on headline generation at the time of publication.

[22] uses an iterative search algorithm similar to hill-climbing for the task of text simplification. Components of the objective function in this model are as follows:

$$f(s) = f_{eslor}(s)^{\alpha} \cdot f_{fre}(s)^{\beta} \cdot (1/f_{len}(s))^{\gamma} \cdot f_{entity}(s)^{\delta} \cdot f_{cos}(s) \tag{2.8}$$

where $f_{eslor}(s)$ is the syntactic log-odds ratio (SLOR) [18] given by a syntax-aware language model, which measures fluency and structural simplicity of a

simplification. SLOR differs from the plain forward-backward language models used in [30][26] in the sense that it penalizes the plain language model probability by length and the product of uni-gram probability of all words in the sentence. This is to factor out the low language model probability of rare words, as they carry important information but oftentimes lead to low language model probability if present in a sentence. SLOR also takes as input the part-of-speech (POS) and dependency tags, claiming better evaluation of language fluency than a plain language model. $f_{fre}(s)$ stands for the Flesch Reading Ease [19] score, which measures the readability of a sentence. The inverse of sentence length ($1/f_{len}(s)$) is designed to give higher scores for shorter sentences. $f_{entity}(s)$ is the count of all named entities, compensating their low language model probabilities. $f_{cos}(s)$ is the cosine distance between the sentence embeddings of the original sentence and the simplified sentence. The edit operations used in this model are both in word and phrase level: removal, extraction, reordering and substitution. At each iteration, each of the edit operation is used to generate multiple candidates, and only those candidates with higher objective scores than the previous step by a threshold would be accepted. Hence, the search algorithm in this model is similar to hill climbing, but have multiple candidate branching at each step. However, the intrinsic issue with only accepting higher score would make the search prone to local optimum. In order to jump out of local optimum, some randomness would be needed to occasionally accept lower score.

UPSA [26] uses simulated annealing (SA) search algorithm for the task of unsupervised paraphrase generation. SA algorithm differs from the hill climbing search in the sense that non-greedy search steps are occasionally allowed. At each iteration one candidate would be generated from one of the word-level operations: insertion, replacement, deletion. If the new candidate state is evaluated to have a higher score than the current state, the current

state is always updated to the candidate; if the new candidate state has a lower score than the current state, SA algorithm would still be possible to accept it with a low probability. Such non-greedy search steps are more likely to be allowed in the beginning of search, and become gradually less likely as the search goes on to settle for a optimum point.

2.3.1 Learning from Search

Despite the state-of-the-art performance in various tasks by the search-based edit models, there are opportunities for enhancement. One major drawback of the search-based framework is that the search process is oftentimes noisy. This is to be expected since the objective functions are manually designed and not learnable.

[24] proposes an iterative search and learn framework for text generation. The core idea is using a Seq2seq model to provide new starting points for the SA search. In the first learning stage, SA algorithm is first performed to search towards the objective function to generate a dataset with pseudo-labels. The generated pseudo labels are used to train a Seq2seq model by cross-entropy loss. In the second learning stage, beam search is performed on the Seq2seq model to obtain a set of candidate outputs. These outputs are used as starting point for SA search again to generate another set of outputs. The Seq2seq is then trained by max-margin loss to maximize the margin between the highest scored instance from these two sets of outputs and the rest. Such alternation between searching (SA) and learning (Seq2seq) is carried on for a couple iterations. Experimental results show that such iterative alternation of searching and learning indeed improve the performance in paraphrase generation task as more iterations are performed, and finally achieve state-of-the-art performance. They attribute the success of this simple approach to the Seq2seq model effectively smoothing out the noise in search results. Moreover, the

positive outcome of such framework shows that learning of search results is feasible. However, learning and searching procedures in this framework both function as black-boxes of input-output correspondence. The natural question along this line would be: Is it also possible to learn from the internal search dynamic?

Attempts are made to use learnable function to improve local search by learning directly from the search dynamics in [5]. This framework learns a function that predicts the outcome of a local search algorithm. In other words, a model predicts the expected maximum objective score seen on a trajectory that starts from a given state x and follows the local search algorithm. To sample from the search dynamics, the search algorithm is first performed to collect search trajectories. Then the learnable evaluation function is trained from the collected trajectories to predict the search outcomes. The training framework consists of two stages running alternately: a search algorithm to maximizes the original objective function; and another search algorithm maximizes the learned evaluation function. Output of either stage would be used as input for the next stage for several iterations. Hence, this framework can be viewed as a smart restarting mechanism for the search algorithm. Experimental results on combinatorial games such as bin-packing, rerouting, and Boolean satisability show promising performance. However, this formulation has yet to demonstrate its capability when it comes to notoriously large search space, such as text generation.

Chapter 3

Methodology

In Section 3.1, we overview the framework of search-based text generation. Specifically, we focus on the paraphrase generation framework of UPSA, which shows state-of-the-art performance in paraphrase generation according to [26] and will serve as our baseline for setting up the foundation of our study. We follow [26] and evaluate our models on the task of unsupervised paraphrase generation.

In Section 3.2, we introduce our proposed models for smoothing the objective function. In particular, we pinpoint the drawback of the current search-based framework, then propose three models for guiding the search process by smoothing the objective function. All three models learn from search trajectories of the SA algorithm.

3.1 Unsupervised Text Generation by Simulated Annealing

Search-based approaches have demonstrated their capability in various unsupervised text generation tasks and have been gaining popularity over recent years [40][22][26]. The search-based framework consists of a manually designed objective function to evaluate the search states with respect to the specified tasks, and a local search algorithm to find the optimum of the objective func-

tion in the sentence space. For the particular task of paraphrase generation, the objective function would need to evaluate grammatical fluency, semantic preservation, and expression diversity. The search algorithm would then perform local search in the sentence space by making word-level edit to search states (i.e. sentence), in order to maximize the objective function. Hence, the task of text generation is formulated as an optimization problem. Details will be provided in the rest of this section to overview components in the objective function to enforce the requirements of paraphrase generation.

3.1.1 Unsupervised Objective Function

The objective function plays the important role of guiding the search algorithm by evaluating the objective score of any given search step. Specifically, the objective function can be considered as a mapping that takes as input an arbitrary sentence x and the context (e.g. original input sentence x_0), and outputs a numerical score indicating the quality of generated sentence x for the given task. In other words, a state with a high score indicates it satisfies most of the requirement for the task.

Language fluency and syntactical coherence are among the most fundamental necessities of language generation. In the neural network regime, fluency and syntax are measured approximately by the probability predicted by a language model. This probability indicates the relative likelihood of a particular sentence is drawn from a specific dataset. More specifically, the language model itself is a probability distribution over the sentence space. The language model is trained to maximize the log-likelihood of each sentence in the dataset. Hence a high probability assigned to a sentence means it was likely drawn from the same distribution that induces the dataset. Since all sentences in the dataset are presumed to be grammatically fluent and syntactically coherent, a high language model probability correlates with higher language fluency and

syntactical coherence:

$$f_{\text{flu}}(\text{x}) = \prod_k P_{\overrightarrow{\text{LM}}}(w_k|w_1, \cdots w_{k-1}) \tag{3.1}$$

where k is the length of sentence $\mathbf{x}$. The language model used in our experiment is a two-layer LSTM model with 300 hidden units. No parallel supervision is used for training the language model.

The main goal of paraphrase generation is to generate text that is distinct from the original input. Hence, sentence with more different wording compared with the original input should be assigned a higher score. The BLEU [33] score is an automatic measure for lexical similarity, which computes the length-penalized n-gram overlaps between two text sequences. Then $1-\text{BLEU}$ would yield higher score for output dissimilar to the input. In this experiment, the lexical dissimilarity of a sentence is measured by $1-\text{BLEU}$ against the original sentence:

$$f_{\text{lex}}(\text{x}|\text{x}_0) = (1 - \text{BLEU}(\text{x}, \text{x}_0))^S \tag{3.2}$$

where S is a hyperparameter controlling the multiplicative weight of the lexical diversity component.

Although different wording is desired, the semantic meaning of the sentence should not change during the paraphrasing process. To enforce this requirement, word-level or sentence-level embedding distance are used to measure the semantic similarity between input and output. In such a way, sentences with similar meaning would have a shorter embedding distance, and vice versa. Such distance metric is not particularly sensitive to any particular word in the sentence as all words are mapped to embeddings. UPSA adopts two ways for measuring semantic similarity: mini-max keyword embedding distance and

sentence embedding distance:

$$f_{\text{sem}}(\mathrm{x}|\mathrm{x}_0) = f_{\text{sem,key}}(\mathrm{x}|\mathrm{x}_0)^P \cdot f_{\text{sem, sen}}(\mathrm{x}, \mathrm{x}_0)^Q \tag{3.3}$$

$$f_{\text{sem,key}}(\mathrm{x}|\mathrm{x}_0) = \min_{e \in \text{keywords}(\mathrm{x}_0)} \max_j \{\cos(\boldsymbol{w}, \boldsymbol{e})\} \tag{3.4}$$

$$f_{\text{sem, sen}}(\mathrm{x}, \mathrm{x}_0) = \cos(\boldsymbol{x}, \boldsymbol{x}_0) \tag{3.5}$$

where P and Q are the relative weight of keyword similarity and sentence similarity, respectively. Here the keywords are extracted by RAKE [39] and the GloVE [34] embeddings are used for computing distance.

The objective function used in UPSA is a multiplicative weighted combination of the aforementioned components. It simultaneously evaluates language fluency, semantic preservation, and lexical diversity:

$$f(\mathrm{x}, \mathrm{x}_0) = f_{\text{flu}}(\mathrm{x}) \cdot f_{\text{sem}}(\mathrm{x}|\mathrm{x}_0) \cdot f_{\text{lex}}(\mathrm{x}|\mathrm{x}_0) \tag{3.6}$$

note that each components are weighted implicitly.

To this end, we have reviewed each individual component in the objective function for paraphrase generation. This objective function specifies the requirement for the task of paraphrase generation. Next to discuss is a search algorithm to maximize such objective function approximately by making word-level local edits.

3.1.2 Simulated Annealing Search

The computational cost of exhaustive search renders it infeasible to perform. The number of possible sentences that can be generated combinatorially grows exponentially with the vocabulary size. In other words, an exhaustive search can be done theoretically, but one would need to face the cost of a branching factor of 30,000 (i.e. vocabulary size). Moreover, for the task of paraphrase generation specifically, exhaustive search is not needed as there could be many plausible paraphrases for a sentence. Being able to find an acceptable solution efficiently should suffice.

The general local search algorithm solves discrete optimization problem by starting from a candidate then iteratively moving to neighbours in the solution space by applying local changes, until some criterion is satisfied or the computational budget is exhausted. In other words, every step in the search trajectory itself would be a potential solution. Hence, the main goal is to search efficiently by making moves that follow a good strategy, so that the trajectory can reach at least one high score state. In this dissertation, we use the simulated annealing algorithm described in [26] as the local search algorithm for solving the optimization problem approximately. Paraphrase generation is a monolingual task that usually have significant overlap between input and output. Hence, the input sentence would serve as a reasonable starting point for the search.

Local search algorithms require a well-defined neighbourhood relation in the solution space for state transition. For the task of paraphrase generation, the solution search space is defined to be all finite length sentences. Search steps are realized by making word-level edit operations. More specifically, each state of the search is represented by:

$$\mathrm{x_t} = (w_{t,i} | i \in [0, l]) \tag{3.7}$$

where t indexes time steps in the search process, and l is the length of the sentence corresponding to state $\mathrm{x_t}$.

Local word-level edits used in this framework are insertion, deletion, and replacement. It can be seen that in theory any arbitrary sentence can be generated from any starting point using these edit operations. Word deletion is straightforward to perform, but word insertion and replacement both require adding appropriate new word to the sentence. The new word added to the sentence are sampled from the same language model used for fluency evaluation. Details will be discussed in rest of this section.

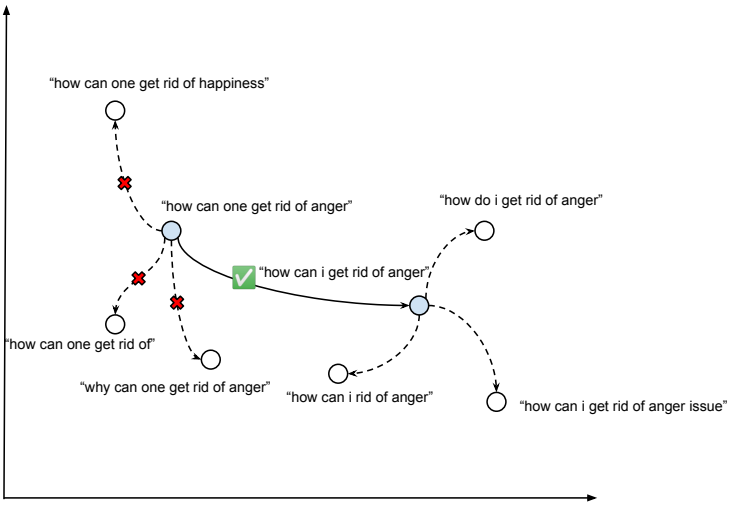


Figure 3.1: An example of one iteration of simulated annealing search. The red crosses represent rejected proposals, while the green swoosh represents accepted proposal.

Simulated annealing (SA) is the choice of search algorithm for maximizing the aforementioned objective function 3.6. The terminology of SA is borrowed from metallurgy, which originally refers to the technique of alternating heating and cooling of a material to produce crystals of desired size while reducing defects. In the context of optimization by stochastic search, this translates to the idea of that the search algorithm would climb to higher function values greedily most of the time, but occasionally allow some non-greedy exploration to escape local optima. SA algorithm explicitly encourages exploration during the initial steps of the search, then becomes more greedy as the search goes on. With this design, the SA algorithm is able to provide reasonable solution even when the search space is large.

One basic iteration of SA for text generation is done by first making edit proposals, then either accepting or rejecting the proposals. For each iteration, a position to be edited i is sampled from a uniform distribution over the length of the sentence, giving each position an equal chance to be edited. Then a neighbouring state x_* is proposed based on the ongoing state x_t at time step t

by performing the one of the aforementioned basic edit operations: insertion, replacement, or deletion, at the sampled position of i:

$$i \sim Uniform(0, l) \tag{3.8}$$

where l is the number of words in sentence $\mathrm{x_t}$.

An edit operation from insertion, deletion, and replacement needs to be chosen once the edit position has been determined. Edit operations are also sampled from the three options. More specifically, given probability of p_{ins}, p_{rep}, p_{del} for insertion, replacement and deletion, respectively, the edit operation is sampled by:

$$z \sim Categorical(p_{ins}, p_{rep}, p_{del}) \tag{3.9}$$

where z is the one-hot vector indicating which edit operation to be performed. In practice, p_{ins}, p_{rep}, p_{del} are set to equal.

If the sampled edit operation is deletion, then it would be straight forward: given the current state $\mathrm{x_t} = (w_{t,1}, \ldots, w_{t,i-1}, w_{t,i}, w_{t,i+1}, \ldots, w_{t,l})$ at step t and i-th word to be deleted, the new candidate sentence becomes $\mathrm{x}_* = (w_{t,1}, \ldots, w_{t,i-1}, w_{t,i+1}, \ldots, w_{t,l})$.

For insertion and replacement, a new word needs to be sampled to add to the sentence. For the appropriate word to be selected, the new candidate word is sampled with probability proportional to the objective score corresponding to the new sentence state induced by the added word

$$p(w_*|\cdot) = \frac{f_{\mathrm{sim}}(\mathrm{x}_*, \mathrm{x}_0) \cdot f_{\mathrm{exp}}(\mathrm{x}_*, \mathrm{x}_0) \cdot f_{\mathrm{flu}}(\mathrm{x}_*)}{Z} \tag{3.10}$$

$$Z = \sum_{w_* \in \mathcal{W}} f_{\mathrm{sim}}(\mathrm{x}_*, \mathrm{x}_0) \cdot f_{\mathrm{exp}}(\mathrm{x}_*, \mathrm{x}_0) \cdot f_{\mathrm{flu}}(\mathrm{x}_*) \tag{3.11}$$

sampling in such way would ensure words that lead to higher scores are more likely to be added. Due to the expensive computational cost of evaluating

Equation 3.10 for every word in the vocabulary, a forward and backward language model is used to truncate this selection to only top-K words. In other words, only the words that lead to high fluency scores are considered. If the new word is to be placed at the i-th slot in sentence x_t, the top-K words being used to evaluate 3.10 are restricted to:

$$\mathcal{W}_t = \text{top-}K_{w_*}[p_{\overrightarrow{\text{LM}}}(w_{t,1}, \ldots, w_{t,i-1}, w_*) \cdot p_{\overleftarrow{\text{LM}}}(w_*, w_{t,i+1}, \ldots, w_{t,l_t})] \tag{3.12}$$

After the new candidate word is proposed, the new candidate state generated by replacement operation would be $\mathrm{x}_* = (w_{t,1}, \ldots, w_{t,i-1}, w_*, w_{t,i+1}, \ldots, w_{t,l})$. Likewise, the new candidate state proposed by insertion operation would be $\mathrm{x}_* = (w_{t,1}, \ldots, w_{t,i-1}, w_{t,i}, w_*, w_{t,i+1}, \ldots, w_{t,l})$.

Copy mechanism is incorporated to preserve named entities and rare key words. These words are usually kept in the sentence during paraphrasing since they carry relatively high amount of important information. However, in practice sentences with these rare words are usually given very low probabilities by language models due to their low uni-gram probabilities. Hence, the practical issue of these rare words is that they are likely to be deleted or replaced during the search process, but are not likely to be recovered again due to the low language model probabilities of sentences having these words. Inspired by [10], the copy mechanism is incorporated in the word sampling process by augmenting the top-k candidates with all words in the input sentence. Hence, it would be easier for the model to copy these words from the original sentence. In particular, the truncated candidates become:

$$\widetilde{\mathcal{W}_t} = \mathcal{W}_t \bigcup \{w_{0,1}, \ldots, w_{0,l_0}\} \tag{3.13}$$

where $\mathcal{W}_t$ is the top-K words truncated by the forward-backward language model in 3.12, and $\{w_{0,1}, \ldots, w_{0,l_0}\}$ are the words in the input sentence x_0.

The SA algorithm decides if the new proposal x_* would be accepted. The essence of SA algorithm is to always accept proposals that lead to higher scores,

but lower scored proposals are occasionally accepted as a means to escape local optima. The time-dependent acceptance ratio is given by:

$$p(\text{accept}|\mathrm{x}, \mathrm{x}_*, \mathrm{x}_0, T) = \min\{1, e^{\frac{f(\mathrm{x}_*|\mathrm{x}_0) - f(\mathrm{x}|\mathrm{x}_0)}{T}}\} \tag{3.14}$$

where T is the temperature parameter controlling how likely non-greedy steps are allowed. T is large at the beginning of search then gradually cools down as search goes on. As shown, the probability of accepting a proposal x_* is always 1 if $f(\mathrm{x}_*|\mathrm{x}_0) > f(\mathrm{x}|\mathrm{x}_0)$. Otherwise, the probability of accepting a new proposal x_* that leads to lower objective score depends on the absolute difference between $f(\mathrm{x}_*|\mathrm{x}_0)$ and $f(\mathrm{x}|\mathrm{x}_0)$. Earlier steps yield high probability to accept a proposal x_* when $f(\mathrm{x}_*|\mathrm{x}_0) < f(\mathrm{x}|\mathrm{x}_0)$ due to the larger T value. The gradual reduction of temperature in Equation 3.14 can be interpreted as the gradual increase in greediness for high objective score. We follow [26] and set initial temperature to $T_0 = 3 \times 10^{-2}$, which is then linearly decreased to zero after a fixed number of search iterations. In this manner, the search algorithm is encouraged to explore a wide region of the search space at the beginning of search. As the search progresses the algorithm commits to an optimum. In theory, with sufficient budget steps for search, the probability of the algorithm settling with a global optimum converges to 1. However, the large search space makes this theoretical guarantee unrealistic. Instead, the main focus of the SA search is to efficiently find a near optimal solution without global inference.

3.2 Learning From Search

One drawback of the search based text generation framework underlies in the manually designed objective function, which is not learnable. Previous work [26][30] show that maximizing the objective function by search indeed improves the evaluation metrics (e.g., BLEU and iBLEU) on average for a dataset. Such evidence show that there is indeed a correlation between the manually

source	what are the things you would like to change if you are given one
	what are that things you would like to change if you are given one
	what are that thing you would like to change if you are given one
	what is that thing you would like to change if you are given one
	what is that one thing you would like to change if you are given one
	what is that one thing you like to change if you are given one
	what is that one thing you want to change if you are given one
	what is that one thing you want to change if are given one
	what is that one thing you want to change if given one
	what is that one thing you want to change in given one
	what is that one thing you want to change in given life
output	what is that one thing you want to change in your life

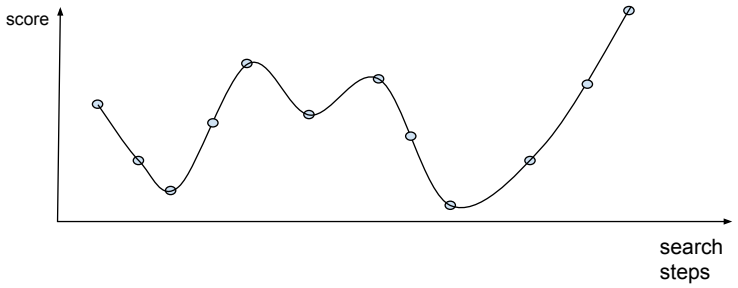


Figure 3.2: An example search trajectory of paraphrase generation.

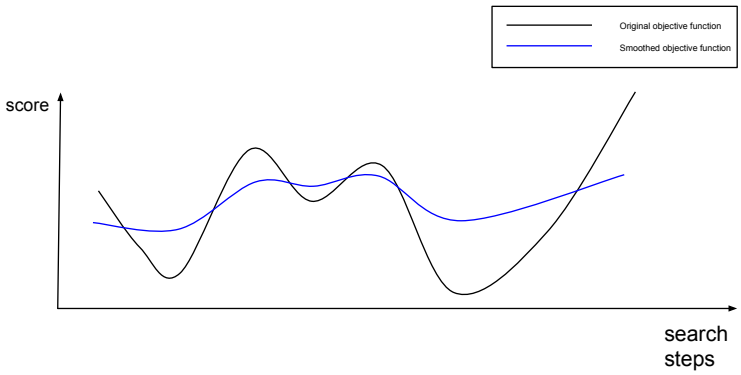


Figure 3.3: A sketch of our motivation to smooth out the original objective function.

designed objective function and the true measure of success on a population level. However, we suspect that the heuristic objective defined with a high level of abstraction may not have the granularity to provide accurate guidance when it comes to each individual sentence. Empirical evidence will be shown to support this claim later in experiments section.

Another challenge of local search in general is the trap of local optima. Being trapped in a local optimum that has a relatively higher score than all of

its immediate neighbours can leave the search stranded for a very long time. Even though SA algorithm would allow occasionally non-greedy move, the probability of this happening only depends on the absolute difference in the objective scores between two states. In our work, we aim to achieve the goal of learning to make sacrifice by making locally non-greedy actions that are beneficial in the long run. This is accomplished by smoothing the heuristic objective function, so that the jumps in between locally optima are more feasible. As demonstrated in Figure 3.3, a smoothed objective function would yield smaller difference among the neighbouring region of a state. Such smaller difference would lead to higher acceptance probability for transition to states with lower scores, but with higher scoring regions around them for the following potential steps to reach.

To this end, we propose three approaches for smoothing out the objective function by learning models from the search process of simulated annealing. The learned models are then combined with the original objective function to form a new smoother approximation of the original objective. The learning task is accomplished by two regression models that try to directly predict the scores, and a Seq2seq model that predicts the likelihood of state transitions implicitly. In the following section, we will go through the proposed models, implementation and model tuning details.

3.2.1 Value Function

Our first approach is to directly train a regression model to approximate the objective function, known as value function [5]. The motivation of this approach is to directly smooth out the objective function numerically. The procedure of this approach is as follows:

1. We perform SA search towards the original objective function, to collect search trajectory samples in the form of $\mathrm{X} = \{\mathrm{x}_i | i \in [0, h]\}$ where i

indexes the time steps in a trajectory, and h is the number of search steps in a search trajectory. Each search state in a trajectory is labeled by its own objective score evaluated by the original objective function f.

2. The value function network is trained to take as input the original sentence x_0, and a state x from a search trajectory, then make a prediction of the objective function score $\hat{f}(\mathrm{x}|\mathrm{x}_0)$ for the given state x. The value function network is trained by mean square error (MSE) against the ground-truth objective scores.

3. For inference, the learned value function is directly combined with the original objective f, yielding a new objective function f_{value}. This new objective function then guide a next iteration of SA search to generate the final output x_T.

The baseline model used in step (1) to collect search trajectory samples are our own re-implementation of UPSA. We evaluate the performance of this re-implementation to validate the baseline. Detailed results are presented in Chapter 4. We faithfully follow all hyperparameter settings as described in [26], setting up a fair comparison between our own models and the original SA search.

In step (2), the value function network $f_v(\mathrm{x}|\mathrm{x}_0)$ is trained to predict the scalar score evaluated by the objective function f. As the backbone of this model, a pre-trained `bert-base-uncased` [6] model is adopted and modified for the regression task. The `bert-base-uncased` model is a transformer [43] model pre-trained on Masked Language Modelling (MLM) and Next Sentence Prediction (NSP) with large scale English data. Pre-training enables the model to learn effective latent representation, which can help relevant downstream tasks. This pre-trained model is capable of performing superbly in a variety of tasks [46] with minimal fine-tuning. Considering this regression task

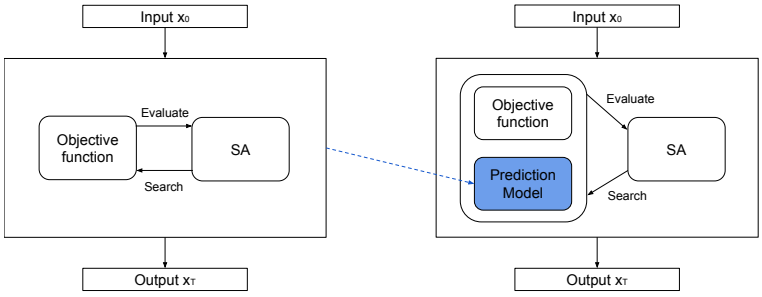


Figure 3.4: An illustration of our proposed learning from search model. Left hand side is the UPSA framework, which generates trajectory samples for training the prediction model(e.g. value function, Seq2seq model). Then the learned prediction model is combined with the original objective function for heuristic evaluation. Search is performed again with the new combined objective function to generate final output.

is largely dependent on Natural Language Understanding(NLU), the BERT model is a reasonable choice for our base model. To modify the pre-trained `bert-base-uncased` model for regression, we extract representation from the pooled output, which is the last layer of the hidden-state of the special leading token of the sequence [CLS]. We then simply add a fully connected layer on top of the pooled output, to generate a scalar prediction of the score. The regression model is then further fine-tuned by mean square error (MSE), in order to minimize the $l2$ error between the score predicted by the value function f_v and the ground truth score given by the heuristic objective function f.

$$J_v(f, f_v) = \sum_{\mathrm{x} \in \mathrm{X}} (f(\mathrm{x}|\mathrm{x}_0) - f_v(\mathrm{x}|\mathrm{x}_0))^2 \tag{3.15}$$

In the inference stage of step (3), the learned value function prediction f_v is directly combined with the original heuristic objective function f by convex combination, yielding an overall objective function f_{value} as

$$f_{value}(\cdot|\mathrm{x}_0) = k \cdot f_v(\cdot|\mathrm{x}_0) + (1-k) \cdot f(\cdot|\mathrm{x}_0) \tag{3.16}$$

3.2.2 Max Value Function

The second approach we propose is a variant of the value function. Instead of directly learning the objective score of a given state, max value function learns and predicts the maximum objective score of a trajectory starting from any particular state. The idea of this approach is similar to smoothing out the objective function with the value function, but we also want to assign a higher scores to states that lead to subsequent high scoring states in the search trajectory. In other words, the goal of the max value function is to learn to sacrifice immediate scores in order to reach even higher scores later in the search trajectory. Max value function is trained and used for inference almost the same way as the value function, except that in step (1), each state is labeled by the maximum objective score that follows that particular state in the search trajectory. Training objective in step (2) is to learn and predict these new pseudo-labels. Inference in step (3) is the same as the value function approach, using a convex combination of the original score f and the new score f_{v*}. We use the exact same `bert-base-uncased` pre-trained model with an added fully connected regression layer on top of the pooled output. Loss function for training the max value function is also mean square error(MSE):

$$J_{v*}(f, f_{v*}) = \sum_{\mathrm{x}\in\mathrm{X}} [f_{v*}(\mathrm{x}|\mathrm{x}_0) - \max_{\mathrm{x}'\in\mathrm{X}} f(\mathrm{x}'|\mathrm{x}_0)]^2 \tag{3.17}$$

here x' is the highest scoring state that follows the particular state x in trajectory X.

3.2.3 Seq2seq Probabilistic Model

The third approach uses a Seq2seq model to smooth out the objective function. This idea is similar to [24], in which they also smooth out the noise in the searching and learning frame work using a Seq2seq model. Unlike the regression models used in the value function and max value function, this approach

uses the emission probability of a Seq2seq model for smoothing effect. More specifically, we train a seq2seq model which takes as input the original input, and predicts the search output (i.e. paraphrases). During inference, the predicted probability of a particular state to be emitted from the seq2seq model given the original input sentence is combined with the original objective function to provide a smoothing effect. The procedure of the seq2seq smoothing approach is as follows

1. We perform SA search towards the original objective function, to collect search trajectory samples $\mathrm{X} = \{\mathrm{x}_i | i \in [0, h]\}$ similar to the value function approaches. Then each trajectory would generate one pseudo-parallel label in the form of $(\mathrm{x}_0, \mathrm{x}_\mathrm{T})$. Each pseudo-parallel training example $(\mathrm{x}_0, \mathrm{x}_\mathrm{T})$ consists of the original input sentence x_0 and the outcome of search x_T (i.e. the final state in search trajectory).

2. The seq2seq model is trained to take the input sentence x_0 of a trajectory and predict the search output x_T. The training objective is to minimize the cross-entropy loss against the pseudo-labels from step (1).

3. For inference, the probability of emitting a particular sentence $P_{i,v}^{(\mathrm{s2s})}$ is directly combined with the original objective f, yielding a new objective function f_{seq2seq}. This new objective function is used by the SA algorithm to search again and output the final sentence x.

To train a Seq2seq model using the pseudo-label from search, we train a state-of-the-art transformer-based Seq2seq model $P_{i,v}^{(\mathrm{s2s})}$ by cross-entropy loss:

$$J_{\mathrm{CE}} = -\sum\nolimits_{i=1}^{l} \sum\nolimits_{v \in \mathcal{V}} w_{i,v} \log P_{i,v}^{(\mathrm{s2s})} \tag{3.18}$$

Where $w_{i,v}$ is the binary value indicating whether the i-th word is v or not in the search output, and $P_{i,v}^{(\mathrm{s2s})} = P_{\mathrm{s2s}}(w_i = v | w_{<i}, \mathrm{x}_0)$ is predicted by the Seq2seq model.

The learned Seq2seq model $P_{(\text{s2s})}$ is combined with the original heuristic objective function f by:

$$f_{\text{s2s}}(\cdot|\text{x}_0) = k \cdot d \cdot P_{\text{s2s}}(\cdot|\text{x}_0) + (1-k) \cdot f(\cdot|\text{x}_0) \tag{3.19}$$

$$P_{\text{s2s}}(\text{x}|\text{x}_0) = \prod_{i=1}^{L} p(w_i|w_{<i}, \text{x}_0) \tag{3.20}$$

Here, d is a scaling factor (set to 100) to scale a probabilities to a similar range of Equation 3.6. k is a relative weighting hyperparameter for the two terms. f_{s2s} becomes the new objective for generating final outputs by another iteration of SA search.

Although both using a Seq2seq model to learn from the pseudo-parallel search output, our work differs from [24], in which P_{s2s} is used to directly generate output tokens in an auto-regressive manner. Instead, our work aims to improve the quality of generated text by improving the original heuristic objective function. Our work provides insight on the search objective, which is an important building block for all search-based text generation frameworks. Moreover, we want to answer the curious question of whether search guided by a heuristic objective function can help improve the objective function in a bootstrapping manner without supervision.

Chapter 4

Experiments

4.1 Dataset

We follow [26] and evaluate our models on unsupervised paraphrase generation with Quora question pair dataset[1], which is a collection of question pairs that are identified as duplicate questions. These duplicated questions can be interpreted as paraphrases for each others. Each data instance consists of its own id, two questions with their respective unique id numbers, as well as a target variable indicating whether they are duplicate questions. Preprocessing of the data involves lower-casing, removing punctuation, and finally deduplicating to avoid any particular instance being assigned more weight than others in training and evaluation. After cleaning up the data, we reserve 10K and 20K of parallel paraphrase pairs for validation and test, respectively. The remaining 500K non-parallel sentences are used for training language model (for fluency score and proposal truncation) as well as collecting search trajectories for training the value function, max value function, Seq2seq model.

[1]Quora question pair dataset`https://www.kaggle.com/c/quora-question-pairs`

4.2 Settings

4.2.1 Search Algorithm Parameters

We faithfully follow the hyperparameter settings of the simulated annealing search algorithm as in UPSA [26] for two reasons: hyperparameters in UPSA are tuned by grid search and validated on the same Quora dataset, which already achieved state-the-art performance in paraphrase generation; we will use this set of hyperparameter settings as the basis of our ablation study, which sets up a fair comparison with the original UPSA model. In particular, initial temperature T_{init} is set to 3×10^{-2}, then decay according to $T = \max\{T_{\text{init}} - C \cdot t, 0\}$, where the annealing rate C is set to 3×10^{-4}. In this way temperature T will drop to zero linearly in 100 iterations. The language model for word proposal and evaluating fluency score f_{flu} is parameterized by a two-layer LSTM model with 300 hidden units. Training of the language model uses only the non-parallel training split of the Quora dataset. The relative weight of each component in the heuristic objective P, Q , and S are set to $8, 1, 1$, respectively.

4.2.2 Search Trajectories Collection

To learn from the SA search dynamics, we perform SA search on the 50K inputs from the training set, generating 50K search trajectories totalling 1.3M search steps for training our proposed models. Each state visited in search trajectories becomes a training data point, which consists of its own trajectory id, search step index, edit operation used, and score evaluated by the original objective. To train the value function f_v, each data instance is cleaned up to keep the original input sentence x_0, the sentence corresponding to the state at the particular search step x_t, and the objective score $f(\mathrm{x}_\mathrm{t})$. For training of max value function f_{v*}, the objective score is set to the highest objective score

in the trajectory after each particular search step. To generate the pseudo-parallel labels for training Seq2seq model P_{s2s}, each search trajectory would contribute exactly one training sample, consisting of only the original input sentence x_0 and the final output sentence x_T from search.

4.2.3 Model Architecture and Tuning

Both of value function f_v and max value function f_{v*} adopt the pre-trained `bert-base-uncased` model by directly adding an extra fully-connected regression layer to the pooled output of the `[CLS]` token, outputting a single scalar prediction. The two models are both further fine-tuned by 50 epochs, with batch size of 64. After tuning with the validation split of the Quora dataset, we choose a scheduler that linearly warms the learning rate up to $1.5e-6$ from 0 in 10 epochs, followed by a cosine decay period.

Our Transformer-based Seq2seq model has 3 layers, 8 heads, and 512 hidden units. The model is trained from scratch by 10 epochs with a batch size of 64, using the same cosine learning rate scheduler as the value function and max value function.

4.3 Evaluation Metrics

To set up a fair comparison with other state-of-the-art methods for unsupervised paraphrase generation, we adopt the standard BLEU and iBLEU as evaluation metrics. BLEU [33] measures length-penalized n-gram overlaps between the output and the ground-truth. iBLEU is a variant of BLEU that explicitly discounts n-gram overlap between the input and output, to favor lexical dissimilarity between input and output in the paraphrase generation task [41]. Both BLEU and iBLEU take value between 0 and 1. A larger BLEU or iBLEU score indicates higher n-gram similarity with the reference. For iBLEU specifically, a higher score also corresponds to higher dissimilarity

Reported in [26]	**BLEU**	**iBLEU**
VAE	13.96	8.16
CGMH	15.73	9.94
SA	18.21	12.03
Our implementations		
UPSA	18.16	12.40
UPSA+Value	18.67	13.11
UPSA+MaxValue	18.81	13.12
UPSA+Seq2SeqProb	**20.20**	**13.42**

Table 4.1: Unsupervised paraphrasing results. BLEU and iBLEU scores are presented in percentage format.

with the input sentence. In the case of a BLEU score of 1, that would mean the generated output is identical to the reference. If the iBLEU is evaluated to be 1, that means the generated output is identical to the reference, while completely different from the input.

4.4 Results of Unsupervised Paraphrase Generation

Table 4.1 presents the result of automatic evaluation for unsupervised paraphrase generation. Experimental results show that our implementation of UPSA reproduces similar performance as reported in [26], verifying that our implementation is correct and setting up the foundation for our study. Moreover, experimental results show that unsupervised paraphrasing by simulated annealing outperforms variational sampling (VAE, [4]) and Metropolis-Hasting sampling (CGMH, [30]) in both BLEU and iBLEU, showing that UPSA is a competitive model for unsupervised paraphrase generation.

It can be seen from Table 4.1 that UPSA boosted by any of the three of our methods consistently outperforms the original UPSA in terms of both BLEU and iBLEU. This shows that our attempt to smoothen the heuristic objective function indeed improved the search performance.

In particular, using the Seq2seq model's probability achieves the best per-

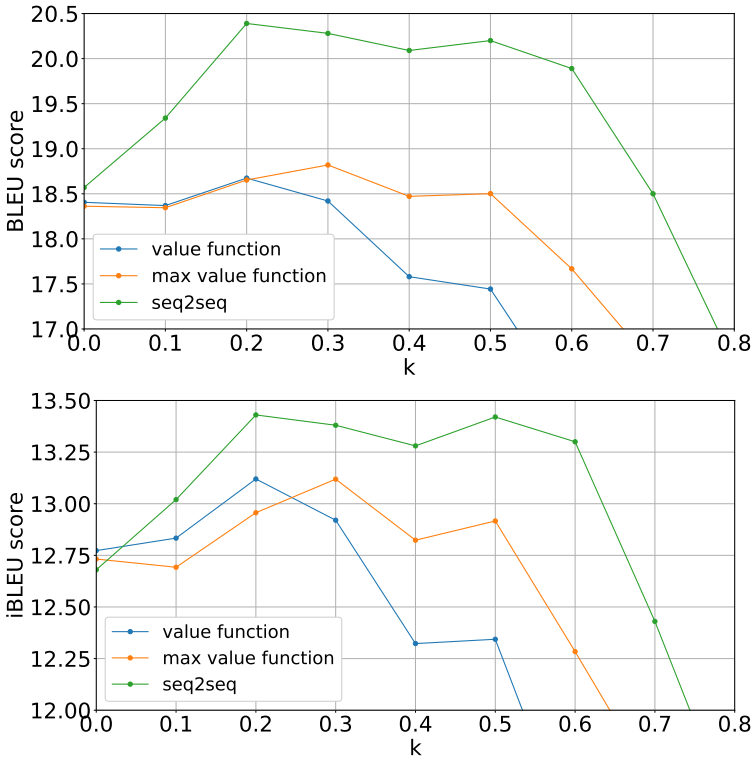


Figure 4.1: BLEU (upper) and iBLEU (lower) versus the relative weight k.

formance among all three of our approaches, yielding an improvement of 2.04 BLEU and 1.02 iBLEU over the original SA search. We attribute this improvement to the learned probability being heterogeneous from the original heuristic objective function, thus providing the strongest smoothing effect.

We further show how the performance varies with the relative weight of the learned models in the search objective in Figure 4.1. As seen, all of our models exhibit a similar trend: the performance increases when the learned models are combined with the objective function f by a small weight. This shows that all of our proposed models are able to learn and model search dynamic to a reasonable extent, and thus improving the search. More specifically, max value function f_{v*} outperforms value function f_v when guiding the search, showing our intuition of learning to sacrifice is indeed effective. However, performance of all three of our models decrease when the learned models are weighted more than the original objective function. Furthermore, if the original objective is

ignored and search is guided only by any of the learn model only, performance drastically decreases. This shows that our learned models are skewed from the original objective function, rendering them inadequate for guiding the search on their own. Note that when combining the learned models with original objective function, weight k for all models and the scaling factor d for Seq2seq are the only tunable hyperparameters. After training our models, they can immediately boost search performance without excessive tuning. This shows our idea of learning to model the search dynamic is effective.

4.5 Analysis

Besides the standard BLEU and iBLEU evaluation metrics, we also include several qualitative and quantitative analyses to demonstrate the insight on what our models learn and why they can improve the search-based text generation framework. These analyses are done by observing the difference in search dynamics with and without the proposed models.

4.5.1 Correlation of Objective Function with Evaluation Metrics

One of the key assumptions for the search-based framework to be feasible is that the manually designed objective function correlates with the true measures of success, BLEU and iBLEU, on a population level. To investigate whether this is indeed true, we measure the point-wise correlation between the heuristic objective and BLEU or iBLEU. Note that our models aim to smooth out the objective function, which intuitively should improve the correlation due to the removal of noise. Hence, we also measure the correlation between the measures of success (i.e., BLEU and iBLEU) and the new objective function given by our value function, max value function, and Seq2seq model.

To get an empirical estimate of the correlation, we randomly sampled 1000 sentence input to perform SA search with original objective function. Each input would generate a search trajectory with multiple states visited. Then each search step in trajectories is re-evaluated by the new objective function given by our learned models. We adopt the Spearman correlation coefficient ρ, which is a rank correlation metric assessing how well a monotonic function can describe the relationship between two variables. This measure of correlation is appropriate for our study since the acceptance probability of proposal depends only on the absolute difference between old and new objective scores. The Spearman correlation between objective scores and BLEU $\rho_{\text{obj,BLEU}}$ is computed as follows:

1. Given a set of search trajectories, we re-evaluate the objective scores and BLEU/iBLEU for each state visited. We then compute the ranking variable rg_{obj} for each objective score, and the ranking variable rg_{BLEU} for each BLEU score.

2. Compute the Pearson correlation coefficient between rg_{obj} and rg_{BLEU} as follows

$$\rho_{\text{obj,BLEU}} = \frac{\text{cov}(\text{rg}_{\text{obj}}, \text{rg}_{\text{BLEU}})}{\sigma_{\text{rg}_{\text{obj}}} \sigma_{\text{rg}_{\text{BLEU}}}} \tag{4.1}$$

   where $\text{cov}(\text{rg}_{\text{obj}}, \text{rg}_{\text{BLEU}})$ is the covariance matrix of the rank variables, and $\sigma_{\text{rg}_{\text{obj}}}$, $\sigma_{\text{rg}_{\text{BLEU}}}$ are standard deviations of the rank variables, respectively.

3. $\rho_{\text{obj,iBLEU}}$ can be computed similarly.

As seen, the Spearman correlation takes a real value in the range of $[0, 1]$. A higher value of ρ indicates more similar ranking between the two variables.

	Correlation between objective scores and BLEU		
Learned model weight	**Seq2seq**	**Value function**	**Max value function**
k=0 (original score)	0.272229	0.272229	0.272229
k=0.1	0.369467	0.266385	0.272365
k=0.2	0.430051	0.259276	0.271866
k=0.3	0.498687	0.250833	0.270353
k=0.4	0.547690	0.240596	0.267475
k=0.5	0.586287	0.227615	0.262661
k=0.6	0.616134	0.210865	0.254570
k=0.7	0.647185	0.188805	0.241569
k=0.8	0.665456	0.159719	0.222038
k=0.9	0.696291	0.122843	0.193834
k=1 (learned model only)	0.713200	0.077553	0.155707

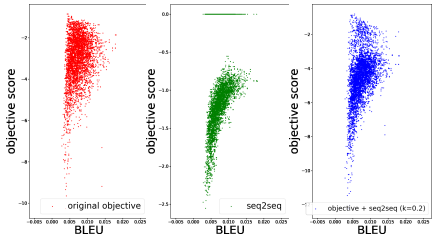
Table 4.2: Correlation between objective scores and BLEU.

	Correlation between objective scores and iBLEU		
Learned model weight	**Seq2seq**	**Value function**	**Max value function**
k=0 (original score)	0.271930	0.271930	0.271930
k=0.1	0.359856	0.266143	0.272092
k=0.2	0.420080	0.259099	0.271619
k=0.3	0.488268	0.250733	0.270139
k=0.4	0.536743	0.240584	0.267302
k=0.5	0.574768	0.227697	0.262534
k=0.6	0.604170	0.211048	0.254493
k=0.7	0.635022	0.189097	0.241544
k=0.8	0.653477	0.160125	0.222065
k=0.9	0.685465	0.123363	0.193915
k=1 (learned model only)	0.704542	0.078181	0.155833

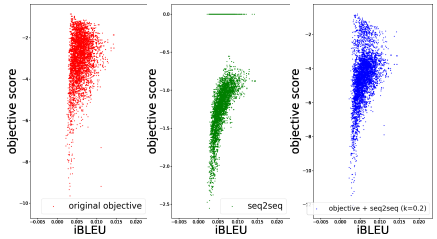
Table 4.3: Correlation between objective scores and iBLEU.

Table 4.2 and 4.3 show the Spearman correlation coefficient of objective function combined with three of our learned models, respectively. It can be seen that in fact the rank correlation between the original objective function and true measures of success is rather low. Moreover, due to the similar formula of BLEU and iBLEU, the two tables present similar patterns: the learned Seq2seq model combined with the original objective function increases correlation with BLEU and iBLEU monotonically as the weight k increases. However, this monotonic increase in correlation with BLEU and iBLEU does not exactly correspond to increase in performance when the new objective function is used. Instead, a small weight given to the learned Seq2seq probability yields the best performance. This shows our Seq2seq model is able to smooth out some noise in the original objective. However, the learned model is not exactly aligned with the original objective function, leading to the completely degenerated performance when it guides the search by itself. On the other hand, the objective function combined with value function and max value function consistently decrease the correlation between objective score and BLEU or iBLEU as the weight k increases. The decline of such correlation does not show a consistent relationship with changes in performance measured by BLEU and iBLEU. This shows that our value function and max value function approaches do not necessarily smooth out the objective function, even though they are still able to improve search performance. However, due to the lack of smoothing effect, such improvement is lower compared to that from the Seq2seq model.

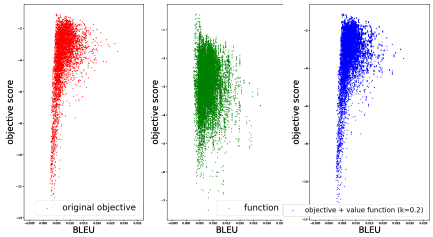
Figure 4.2 presents visualization of the mapping between objective scores and measure of success (BLEU and iBLEU). We can see that the Seq2seq model smooth out the objective function by skewing the objective scores. However, the value function and max value function does not seem to have such an effect. Specifically, from Figure 4.2a and Figure 4.2b we observe that the Seq2seq model combined with the original objective function would pull the



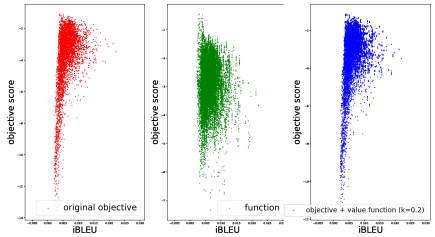
(a) BLEU with Seq2seq model



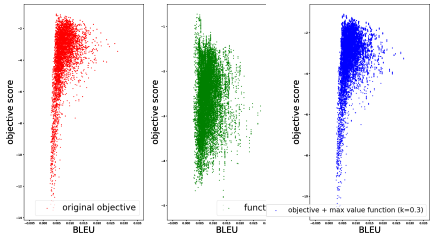
(b) iBLEU with Seq2seq model



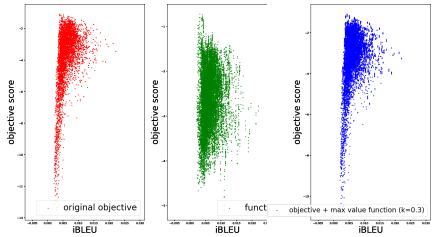
(c) BLEU with value function



(d) iBLEU with value function



(e) BLEU with max value function



(f) iBLEU with max value function

Figure 4.2: visualization of the mapping between objective scores (vertical axes) and measure of success (BLEU and iBLEU, horizontal axes). Red plots are the original score; green plots are score predicted by the value function, max value function, or Seq2seq model; Blue dots are new objective score combined with corresponding model.

objective score down at the lower BLEU or iBLEU region, meaning that the learned model effectively lowers the objective score of some falsely high scored instances when the measure of success (BLEU and iBLEU) is in fact very low. It can also be seen from Figure 4.2a and Figure 4.2b that the learned Seq2seq model seems to skew the objective score towards a monotonically increasing linear function between the objective score and measure of success (BLEU and iBLEU), which is consistent with our previous conclusion that the learned Seq2seq model increases the linear correlation of objective score with BLEU and iBLEU. The value function and max value function, on the other hand, both learn a more flat-tailed Gaussian-like mapping between the objective score and the BLEU or iBLEU.

4.5.2 Acceptance Ratio

All three of our models play the role of "surrogate" objective function, which governs the search by changing the acceptance probability of given proposals. Hence, we investigate how the acceptance ratio changes with the new objective function modified by our proposed models. Furthermore, we are interested in whether there is a quantitative relationship between the acceptance ratio and the measures of success (i.e., BLEU and iBLEU).

To measure the acceptance ratio, we randomly sampled 1000 input sentence from the test set, then perform search with the objective function combined with Seq2seq, value function, and max value function with various weight. An empirical estimate of proposal acceptance ratio is done by counting the average number of accepted proposals out of 100 sampling steps for each model, which is equivalent to the trajectory length given fixed sampling steps. Table 4.4 presents the trajectory length per 100 sampling steps for objective function combined with Seq2seq model, value function, and max value function, respectively with varying weight.

	Trajectory length (per 100 sampling steps)		
Learned model weight	**Seq2seq**	**Value function**	**Max value function**
k=0 (original score)	21.1071	21.1071	21.1071
k=0.1	21.2125	18.1641	18.1611
k=0.2	21.2688	17.4694	17.3814
k=0.3	21.3222	17.0991	17.2432
k=0.4	21.4179	17.4084	16.7757
k=0.5	21.6449	17.4724	16.3543
k=0.6	21.9042	17.7417	17.3984
k=0.7	22.2926	18.5505	18.1631
k=0.8	23.2326	20.006	20.3043
k=0.9	24.4818	22.028	23.5806
k=1 (learned model only)	24.0088	21.9016	27.2828

Table 4.4: Trajectory length (per 100 sampling steps).

As seen, when searching towards the adjusted objective functions, increasing the weight of Seq2seq model would increase the average acceptance ratio consistently, indicating more edit operations are being realized leading to more diverse output. Different from the Seq2seq model, both the value function and max value function show patterns of decreasing acceptance ratio only when given a small weight ($k < 0.7$), but acceptance ratio increases when weight is large ($k > 0.8$). Note that for all three models the optimal weight k for the best BLEU and iBLEU is in the range of $[0.2, 0.3]$. With a weight in this range Seq2seq roughly preserve the same acceptance ratio of the original objective function, while both value function and max value function decreases the acceptance ratio. This shows that the Seq2seq model when combined with the original objective function does not blindly increase of decrease the acceptance probability on average, but rather makes more informed selection when evaluating acceptance probability. The two regression models, on the other hand, are keen on keeping the original content in the sentence.

4.5.3 Chances of Escaping Local Minimum

One of the motivations behind our work is SA algorithm can only escape local optimum by randomness at the initial high temperature stage. We speculate smoothing the heuristic objective can improve search by eliminating the gaps between nearby local optima in the original objective function. Moreover, the max value function is designed to encourage sacrificing short term gain for long term payoffs. This is done by assigning a high score for states that have a low score, but would later on lead to high scoring states. Hence, we investigate how our models change the frequency of escaping local optima that are in the original objective function.

We first perform SA search with objective function modified by value function, max value function, and Seq2seq model, respectively. Then, the collected search trajectories are re-evaluated by the original objective function to check if any local optima are skipped. To identify escapes from local optimum, we find all local minima in search trajectories by re-evaluating the original objective score for every three consecutive steps. A step with lower objective score than its two neighbours is considered to be a local minimum. We are specifically interested in whether the search algorithm can learn to sacrifice by stepping into lower score state that later escape to higher score state than even before the local minimum. Hence, only the escapes from local minima that lead to higher score state would be counted.

Table 4.5 presents the number of escapes from local optima per 100 sampling steps for all three proposed models with varying weights. It can be seen that all three models consistently decreases number of escapes from local minimum on average when the weight k increases, which is contrary to our intuition. Perhaps a reasonable explanation would be the learned models learn to identify local minima in the objective function, thus guiding the search

	Number of escapes from local optimum (per 100 steps)		
Learned model weight	**Seq2seq**	**Value function**	**Max value function**
k=0 (original score)	3.5207	3.5207	3.5207
k=0.1	3.5132	3.3325	3.2544
k=0.2	3.5498	3.1663	3.1523
k=0.3	3.4494	3.1063	3.0562
k=0.4	3.4241	3.1653	2.8991
k=0.5	3.3561	3.1203	2.7539
k=0.6	3.2329	2.8180	2.7269
k=0.7	3.0843	2.7740	2.7329
k=0.8	2.6301	2.5537	2.2895
k=0.9	1.6184	2.1003	1.5397
k=1 (learned model only)	0.6829	1.9566	1.3754

Table 4.5: Number of escapes from local optimum (per 100 steps).

algorithm away from even stepping into local minima.

Chapter 5

Conclusion

Text generation has been an increasingly trending research area in the field of natural language processing. Natural languages are diverse, complicated, and oftentimes syntactically ambiguous, posing challenge for machines to model natural languages over the years. The emergence of deep neural networks have enabled computer systems to understand, process, and generate complicated natural language. Typically, such deep neural network-based text generation models are trained from parallel corpora by maximizing the likelihood of generating the correct output given the input. Nonetheless, there are numerous scenarios where parallel corpora is not available, calling for unsupervised text generation approach that can generate natural language without the need of parallel supervision. One trending approach to unsupervised text generation is by stochastic search towards a manually designed objective function, which evaluates language fluency, semantic meaning, and other task specific attributes. Such objective function is to be maximized by a local search algorithm that navigates the solution space by starting from the input sentence, then performing word-level or phrase-level edit operations including insertion, replacement, and deletion. Search-based approaches have demonstrated their capability in a variety of tasks, including paraphrase generation, text simplification and keyword-to-text generation.

One of the major drawbacks of search-based text generation models is their performance largely depends on the design of the objective function. Oftentimes, the objective function is heuristically designed to specify desired attributes on a high abstraction level. Such design of objective function is shown to correlate with the true measure of success (BLEU and iBLEU) on a population level, but potentially lack granularity when it comes to each single sentence. Moreover, due to the complex components in the objective function, the optimization landscape is likely to be not smooth, posing significant challenge for the search algorithm to find the optimal solution.

In this dissertation, we address the research question of smoothing and improving the objective function that guides the search. To accomplish this, preliminary search is performed on a given task to collect sample search trajectories. Then we propose three deep neural network-based models to learn and model the search dynamic using the collected search trajectories. Finally, the learned models would be combined with the original objective function to guide a next iteration of search to generate final output.

Experimental results on unsupervised paraphrase generation task with Quora question pairs dataset show all three of our proposed models are indeed capable of improving paraphrase generation performance by adjusting only the objective function. More in-depth analyses show that our three models in fact lead to different search behavior, while all are able to improve the performance in terms BLEU and iBLEU.

Based on what we observed and learned from our study, we identify the follow directions to explore for future work:

Iterative Search and Learn: one direct increment from this dissertation is to have a iterative bootstrapping update between searching and learning. Specifically, one can generate and collect search trajectories using one of our proposed models, which can be used to train a second iteration of value func-

tion, max value function or Seq2seq model. The newly learned model can be combined into the objective function again and the process goes on. Previous work [24] have shown that a similar bootstrapping alternation between searching and learning using a Seq2seq model to generate new starting points for search can indeed improve search performance over each iteration. It would be interesting to find out if an iterative searching and learning of the search dynamic can leads to improvement in search.

Adversarial Training: this idea would only be realistic if a learned model itself suffices for guiding the search, which our models currently don't. However, a sketch of adversarial learning would make sense if such fully learnable objective function exists: the generator in the adversarial framework is simply the search algorithm or the candidate generator, whose goal would be to generate good quality text (e.g. paraphrases); the discriminator plays the role of objective function, whose goal is to differentiate if a search state is acceptable (e.g. if the new state is a good paraphrase). However, the specific training scheme needs to be deliberated to avoid both the generator and discriminator drift too far away from the original objective.

Embedding Search: one major limitation of many search-based frameworks is the primitive word-level edit operations: editing text in such a way would require multiple steps to realize large structural modification. However, this could be potentially hard to achieve since most local search algorithms do not explicit retain a history of edits. Hence, if search is performed in the embedding space, one would speculate the underlying sentence structure can change more consistently. The potential challenge of embedding space search is that a tiny shift in numerical embedding space may lead to unchanged discrete text after mapping back to the word space. However, it is still tempting to search by editing a more global representation of text.

References

[1] F. Alva-Manchego, J. Bingel, G. Paetzold, C. Scarton, and L. Specia, "Learning how to simplify from explicit labeling of complex-simplified text pairs," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017. [Online]. Available: https://aclanthology.org/I17-1030. 20, 21

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. 9, 16

[3] D. H. Ballard, "Modular learning in neural networks.," in *AAAI*, vol. 647, 1987, pp. 279–284. 22

[4] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, 2016. [Online]. Available: https://www.aclweb.org/anthology/K16-1002. 3, 22, 49

[5] J. Boyan and A. W. Moore, "Learning evaluation functions to improve optimization by local search," *Journal of Machine Learning Research*, pp. 77–112, 2001, ISSN: 1532-4435. [Online]. Available: https://dl.acm.org/doi/10.1162/15324430152733124. 29, 40

[6] J. Devlin, M.-W. Chang, K. Lee, and K. 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*, 2019, pp. 4171–4186. [Online]. Available: https://www.aclweb.org/anthology/N19-1423. 6, 17, 41

[7] Y. Dong, Z. Li, M. Rezagholizadeh, and J. C. K. Cheung, "EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3393–3402. [Online]. Available: https://www.aclweb.org/anthology/P19-1331. 20

[8] E. Goldberg, N. Driedger, and R. I. Kittredge, "Using natural-language processing to produce weather forecasts," *IEEE Expert*, vol. 9, no. 2, pp. 45–53, 1994. 9

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016. 12

[10] J. Gu, Z. Lu, H. Li, and V. O. Li, "Incorporating copying mechanism in sequence-to-sequence learning," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016. [Online]. Available: `https://aclanthology.org/P16-1154`. 37

[11] J. Gu, C. Wang, and J. Zhao, "Levenshtein transformer," in *Advances in Neural Information Processing Systems*, 2019. [Online]. Available: `shorturl.at/wQ123`. 21

[12] M. El-Haj, A. AbuRa'ed, M. Litvak, N. Pittaras, and G. Giannakopoulos, "The financial narrative summarisation shared task (FNS 2020)," in *Proceedings of the 1st Joint Workshop on Financial Narrative Processing and MultiLing Financial Summarisation*, 2020. 9

[13] S. Havens and A. Stal, *Use bert to fill in the blanks*, 2019. [Online]. Available: `https://github.com/Qordobacode/fitbert`. 18

[14] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998. 15

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 15

[16] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989. 12

[17] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International conference on machine learning*, PMLR, 2015, pp. 2342–2350. 15

[18] K. Kann, S. Rothe, and K. Filippova, "Sentence-level fluency evaluation: References help, but can be spared!" In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, 2018. [Online]. Available: `https://aclanthology.org/K18-1031`. 26

[19] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," Naval Technical Training Command Millington TN Research Branch, Tech. Rep., 1975. 27

[20] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. 22

[21] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *Advances in neural information processing systems*, 2015, pp. 3294–3302. 24

[22] D. Kumar, L. Mou, L. Golab, and O. Vechtomova, "Iterative edit-based unsupervised sentence simplification," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7918–7928. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.707. 4, 26, 30

[23] M. La Quatra and L. Cagliero, "End-to-end training for financial report summarization," in *Proceedings of the 1st Joint Workshop on Financial Narrative Processing and MultiLing Financial Summarisation*, 2020. [Online]. Available: https://aclanthology.org/2020.fnp-1.20. 9

[24] J. Li, Z. Li, L. Mou, X. Jiang, M. Lyu, and I. King, "Unsupervised text generation by learning from search," in *Advances in Neural Information Processing Systems*, 2020, pp. 10 820–10 831. [Online]. Available: shorturl.at/djmpP. 28, 43, 45, 62

[25] Z. Li, X. Jiang, L. Shang, and H. Li, "Paraphrase generation with deep reinforcement learning," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 3865–3878. [Online]. Available: https://www.aclweb.org/anthology/D18-1421. 4

[26] X. Liu, L. Mou, F. Meng, H. Zhou, J. Zhou, and S. Song, "Unsupervised paraphrasing by simulated annealing," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 302–312. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.28. 6, 27, 30, 34, 38, 41, 46,

[27] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. [Online]. Available: https://aclanthology.org/D15-1166. 9

[28] J. Lyons, *Natural Language and Universal Grammar: Volume 1: Essays in Linguistic Theory*. Cambridge University Press, 1991. 2

[29] E. Malmi, S. Krause, S. Rothe, D. Mirylenka, and A. Severyn, "Encode, tag, realize: High-precision text editing," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 2019, pp. 5054–5065. [Online]. Available: https://www.aclweb.org/anthology/D19-1510. 21

[30] N. Miao, H. Zhou, L. Mou, R. Yan, and L. Li, "CGMH: Constrained sentence generation by metropolis-hastings sampling," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019, pp. 6834–6842. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4659. 22–25, 27, 38, 49

[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. 14

[32] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," 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. [Online]. Available: https://aclanthology.org/N18-1049. 26

[33] K. Papineni, S. Roukos, T. Ward, and W.-J. 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. [Online]. Available: https://www.aclweb.org/anthology/P02-1040. 7, 32, 48

[34] J. Pennington, R. Socher, and C. D. 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. 14, 33

[35] R. Puduppully, L. Dong, and M. Lapata, "Data-to-text generation with content selection and planning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 6908–6915. 9

[36] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. 18

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. 18

[38] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," *Transactions of the Association for Computational Linguistics*, 2020. 18

[39] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text Mining: Applications and Theory*, pp. 1–20, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470689646.ch1. 33

[40] R. Schumann, L. Mou, Y. Lu, O. Vechtomova, and K. Markert, "Discrete optimization for unsupervised sentence summarization with word-level extraction," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 5032–5042. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.452/. 4, 25, 30

[41] H. Sun and M. Zhou, "Joint learning of a dual SMT system for paraphrase generation," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2012. [Online]. Available: https://aclanthology.org/P12-2008. 7, 48

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf. 2, 6, 15

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017. [Online]. Available: shorturl.at/dBGX0. 3, 17, 41

[44] S. Wiseman, S. Shieber, and A. Rush, "Challenges in data-to-document generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017. [Online]. Available: https://aclanthology.org/D17-1239. 9

[45] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016. 9

[46] P. Xia, S. Wu, and B. Van Durme, "Which *BERT? A survey organizing contextualized encoders," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020. [Online]. Available: https://aclanthology.org/2020.emnlp-main.608. 41

[47] R. Zellers, Y. Bisk, R. Schwartz, and Y. Choi, "Swag: A large-scale adversarial dataset for grounded commonsense inference," *arXiv preprint arXiv:1808.05326*, 2018. 18

[48] X. Zhang and M. Lapata, "Sentence simplification with deep reinforcement learning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2017. [Online]. Available: http://aclweb.org/anthology/D17-1063. 21

[49] S. Zhao, R. Meng, D. He, A. Saptono, and B. Parmanto, "Integrating transformer and paraphrase rules for sentence simplification," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2018. [Online]. Available: https://aclanthology.org/D18-1355. 19

[50] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 19–27. 17