Discovering Reward Functions for Language Models

by

Yongchang Hao

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

Master of Science

Department of Computing Science
University of Alberta

Abstract

Language models are a fundamental component of natural language processing (NLP) systems. Numerous successful deployments of modern artificial intelligence systems are based on language models, including GPT-4 and ChatGPT. In practice, they are often trained with the teacher forcing objective, where the goal is to predict the next token given the context. One problem with this objective is that it suffers from inconsistency between training and inference, which leads to compounding errors during the inference phase. Reinforcement learning (RL) is considered a promising approach to address these issues. However, it is usually difficult to design reward functions for language models. Previous attempts at designing reward functions are typically sparse and handcrafted for a specific task only.

In this thesis, we propose a task-agnostic approach that derives a step-wise reward function directly from a model trained with the teacher forcing objective. Our work shows that under a common assumption, language models are trained to implicitly capture the rewards. This simple connection allows us to derive a reward function from any pre-trained language models. The derived reward functions can be used to conduct reinforcement learning for language modeling. We conduct experiments on different sequence-to-sequence tasks, including dialogue and paraphrase generation. Empirical results show that the model trained with our reward function outperforms self-training and reward regression methods on both tasks, confirming the effectiveness of our derivation.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Lili Mou, for his guidance and support throughout my master's study. Together, we discovered an interesting connection between language models and reward functions, which is the central idea of the thesis. I am grateful for his patience and encouragement along the way.

As a supervisor, Dr. Lili Mou is always supportive and understanding. He is always willing to listen to my ideas and provide constructive feedback. Over the past two years, I have learned a lot from him, from research methodology to academic writing. I am also grateful for his help in my personal development. It is no exaggeration to say that the work would not be published without his help. Overall, Dr. Lili Mou is an excellent supervisor and a great mentor. I am very fortunate to have him as my supervisor.

I would also like to thank my parents for their unconditional love. Due to the pandemic, I have not been able to visit them for more than two years. I am grateful for their understanding and encouragement. They are always there for me when I need them, and strongly support me in my academic journey.

The research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant No. RGPIN2020-04465, the Amii Fellow Program, the Canada CIFAR AI Chair Program, a UAHJIC project, a donation from DeepMind, and the Digital Research Alliance of Canada (alliancecan.ca).

Table of Contents

List of Tables

List of Figures

Abbreviations

AI stands for artificial intelligence.

GPT stands for generative pre-trained Transformer.

IRL stands for inverse reinforcement learning.

LM stands for language model.

ML stands for machine learning.

NLP stands for natural language processing.

RL stands for reinforcement learning.

Chapter 1

Introduction

1.1 Motivation

Language models have been widely used in natural language processing (NLP) tasks, such as machine translation [1], dialogue generation [2], and summarization [3]. The goal of a language model is to learn the probability distribution of a sequence of words. With the rise of deep learning, this is usually achieved by training a neural network. In recent years, the Transformer [1] has become the dominant architecture for language modeling. The Transformer is a self-attention based model that can capture long-range dependencies in a sequence and compute in parallel. It has achieved state-of-the-art performance on many NLP tasks [4, 5].

Such models are often trained to predict the next token given the context. This is known as teacher forcing. It has been widely applied in multiple literature [1, 6, 7]. However, there are two main issues with this practice:

- Teacher-forcing training is data-hungry because parallel datasets are usually expensive to obtain. On the other hand, there are numerous unlabeled, non-parallel datasets available. This poses an urge to efficiently exploit non-parallel data.

- Teacher forcing introduces a discrepancy between training and inference because the model learns to predict the next word based on the partial groundtruth

reference during training, whereas in inference the model predicts the next word based on its self-generated previous words. This undesired discrepancy is known as *exposure bias* [8–11].

1.2 Main challenges

Several methods have been proposed to address the problems above individually.

For the first problem, a straightforward method is to generate pseudo-parallel sentences for data augmentation, such as self-training [12], sequence-level knowledge distillation [13], and back-translation [14]. However, the exposure bias remains in such cases.

To address the second problem, the model should consider its self-generated sentences during training. Common solutions are often based on reinforcement learning (RL). In text generation, however, there does not exist a naturally defined reward function for RL. Researchers have proposed various heuristic scores as the reward, such as BLEU for translation [15] and ROUGE for summarization [16]. These reward functions are task-specific and not generalizable to other tasks. Further, these rewards require parallel data, failing to address the first problem; they are typically sparse (only non-zero at the end of a sentence), making RL training difficult.

As seen, there is no existing method that can address both problems simultaneously. The goal of this work is to address both in one framework with a learned, dense reward function that does not require human preference annotations.

1.3 Contributions of this thesis

In this thesis, we propose a novel framework to address the above two problems simultaneously. Specifically, we reveal that language models trained with teacher forcing are naturally reward functions. We explain the equations to induce a reward function for RL training. With the equation, we can easily induce a reward function

from an off-the-shelf language model without any parameter tuning.

Our method is task-agnostic and does not require handcrafted engineering or heuristics. Further, our reward function provides dense (step-wise) training signals, which makes RL training much easier than sparse rewards. Additionally, the reward function derived from the language model does not directly participate in the generation, which allows the model to explore based on its own prediction and thus alleviates the exposure bias.

We conduct experiments on dialogue generation and paraphrase generation. We first train a language model with teacher forcing and then apply RL on the same model based on our induced reward function with unlabeled data. The empirical results suggest that our method leads to better performance compared with several baselines, including self-training and task-specific heuristic reward learning, on both tasks. This confirms the effectiveness and generality of our framework.

To summarize, the main contributions of this thesis are as follows:

- We propose a novel framework to address the two problems of teacher-forcing training simultaneously. Our framework is task-agnostic and does not require handcrafted engineering or heuristics.
- We reveal that language models trained with teacher forcing are naturally reward functions. We give out the equations to induce a reward function for RL training.
- We conduct experiments on dialogue generation and paraphrase generation. The empirical results suggest that our method leads to better performance compared with several baselines on both tasks.

1.4 Thesis structure

In this chapter, we introduce the motivation and contributions of this thesis. The rest of the thesis is organized as follows:

- Chapter 2 introduces the background of this thesis, including the Transformer, language modeling, and reinforcement learning. In particular, we start with the history of language models. We illustrate the development of language models from the traditional n-gram models to the modern neural language models, with a focus on the Transformer models. In addition, we will also introduce the background of reinforcement learning, including the Markov decision process and policy gradient. The chapter ends with a brief literature review. The details are postponed to later chapters.

- Chapter 3 describes the detailed derivation of the reward function from the language model. We first present the formulation of the language modeling task as a Markov decision process. Then, we present the common assumption that we borrow from the inverse reinforcement learning literature. We show our main argument that the language model trained with teacher forcing is a reward function. The chapter ends with the overall algorithm of our framework.

- Chapter 4 presents the empirical results of our framework on dialogue generation and paraphrase generation. We first introduce the datasets and evaluation metrics. Then, we present the experimental results and analysis. We also conduct ablation studies to verify the effectiveness of our framework.

- Finally, Chapter 5 concludes this thesis. We discuss the limitations of our framework. Moreover, we provide an outlook for potential future work arising from our framework.

Chapter 2

Background

In this chapter, we will first illustrate language models in Section 2.1. We will show the development of language models with a focus on neural network-based ones, starting with the recurrent neural network. We will see the origin of the attention mechanism in recurrent models, which leads to the well-known Transformer models. We will finish this section with a discussion of two different branches of autoregressive Transformers: the encoder–decoder and decoder-only architectures.

In Section 2.2, we will introduce the concepts in reinforcement learning. We will see how language modeling is depicted in the framework of the Markov decision process, the foundation of reinforcement learning. We will show policy gradient methods are well-suited for language model training. We finish this section with a discussion of different approaches to text generation beyond language modeling with teacher forcing.

2.1 Language models

Language models have been extensively studied in the past few decades. The underlying mathematical foundation was first developed by Markov [17], who introduced the concept of n-gram. The n-gram models are based on the Markov assumption, which assumes that the probability of a word only depends on the previous $n-1$ words. Shannon [18] applied the n-gram model to approximate the probability of

English sentences.

The n-gram models are simple and efficient, but they suffer from the curse of dimensionality. Specifically, the number of possible n-grams grows exponentially with the size of the vocabulary.

The turning point of language models is the introduction of neural networks proposed by Bengio and Bengio [19], aimed to solve the curse of dimensionality problem. However, they only used simple feed-forward neural networks, which are incompetent to model longer sentences. In the following years, recurrent neural networks were introduced to model longer contexts. Recently, more advanced models like Transformer [1] have achieved great success.

2.1.1 Definition

Language models are used to estimate the probability of sentences. The probability of a sentence is the product of the conditional probability of each word given the previous words in the sentence. Building a language model is essentially modeling the conditional probability of the next word given the previous words.

Formally, given a sentence $\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$, the probability of the sentence is defined as:

$$P(\boldsymbol{y}) = P(y_1) \prod_{t=2}^{T} P(y_t | y_1, y_2, \ldots, y_{t-1}) \tag{2.1}$$

$$\approx \hat{P}_{\mathrm{LM}}(y_1) \prod_{t=2}^{T} \hat{P}_{\mathrm{LM}}(y_t | \boldsymbol{y}_{<t}), \tag{2.2}$$

where $\boldsymbol{y}_{<t} = (y_1, y_2, \ldots, y_{t-1})$ is the previous words of the word y_t. The language model $\hat{P}_{\mathrm{LM}}(y_t | \boldsymbol{y}_{<t})$ outputs the conditional probability for a given word y_t based on the previous words $\boldsymbol{y}_{<t}$.

2.1.2 N-gram language models

The n-gram language models are based on the Markov assumption, which assumes that the probability of a word only depends on the previous $n-1$ words. The n-gram

language models are defined as:

$$\hat{P}_{\mathrm{LM}}(y_t|\boldsymbol{y}_{<t}) = \begin{cases} P(\boldsymbol{y}_{\leq t}) & \text{if } t \leq n \\ P(y_t|y_{t-n}, y_{t-n+1}, \ldots, y_{t-1}) & \text{if } t > n, \end{cases} \tag{2.3}$$

where we simply cut off the early prefix of the sentence if the length of the sentence is greater than n.

To train such a model, we need to estimate the probability of each n-gram, which can be achieved by counting the number of occurrences of the n-gram in the training data and normalizing it by the number of occurrences of the previous $n-1$ words. For example, the probability of the word y_t given the previous $n-1$ words $\boldsymbol{y}_{<t}$ can be estimated as:

$$\hat{P}_{\mathrm{LM}}(y_t|\boldsymbol{y}_{<t}) = \frac{C(y_{t-n+1}, y_{t-n+2}, \ldots, y_{t-1}, y_t)}{C(y_{t-n+1}, y_{t-n+2}, \ldots, y_{t-1})}, \tag{2.4}$$

where $C(\cdot)$ is the counting function given the training data.

The n-gram language models are simple and intuitive. However, they suffer from the curse of dimensionality. For example, if we have a vocabulary of size V, the number of possible n-grams is V^n. It is usually infeasible to estimate the probability of all possible n-grams given the limited amount of training data. This causes the n-gram language models to perform poorly on unseen n-grams because the probability of unseen n-grams is always zero.

2.1.3 Neural network-based language models

The feed-forward neural network for language modeling

The neural network-based language models are first introduced by Bengio and Bengio [19]. They used a simple feed-forward neural network to model the conditional probability of the next word given the previous words. The neural network-based language models are defined as:

$$\hat{P}_{\mathrm{LM}}(y_t|\boldsymbol{y}_{<t}) = \mathrm{softmax}(W_o h_t + b_o), \tag{2.5}$$

where W_o and b_o are the output weight matrix and bias vector. The vectorial representation h_t, also known as the hidden state, is computed as:

$$h_t = \tanh(W_h f(\boldsymbol{y}_{<t}) + b_h), \tag{2.6}$$

where f is the feature function that maps the previous words to a vectorial representation. The matrix W_h and vector b_h are the hidden weight matrix and bias vector for the input layer.

Although the feed-forward neural network language models solve the curse of dimensionality problem, they suffer from the long-term dependency problem. Specifically, the feed-forward neural network language models can only access the previous words within the sliding window and ignore all the other words.

The recurrent neural network for language modeling

The recurrent neural network is then proposed to be the backbone of language models as it is designed to handle sequential data [20]. The general idea is to capture the information of all the previous words through the hidden state. The hidden state is updated at each time step and is passed to the next time step. The recurrent neural network-based language models are defined as:

$$h_t = f(h_{t-1}, y_{t-1}), \tag{2.7}$$

$$\hat{P}_{\mathrm{LM}}(y_t|\boldsymbol{y}_{<t}) = \mathrm{softmax}(W_o h_t + b_o), \tag{2.8}$$

where f is the recurrent function that updates the hidden state h_t based on the previous hidden state h_{t-1} and the previous word y_{t-1}. The parameters W_o and b_o are the output weight matrix and bias vector.

The choice of f is crucial for the performance of the recurrent neural network. The straightforward choice is the simple recurrent unit [21], defined as:

$$h_t = \tanh(W_{xh} y_{t-1} + W_{hh} h_{t-1} + b_h). \tag{2.9}$$

The simple recurrent unit suffers from the vanishing gradient problem [22], caused by the repeated multiplication of the weight matrix W_{hh} in the recurrent function.

To address this issue, a common choice is the long short-term memory [23]. It enhances the hidden state h_t by maintaining another variable c_t. The whole function f is formally defined as:

$$i_t = \sigma(W_{xi} y_{t-1} + W_{hi} h_{t-1} + b_i), \tag{2.10}$$

$$f_t = \sigma(W_{xf} y_{t-1} + W_{hf} h_{t-1} + b_f), \tag{2.11}$$

$$o_t = \sigma(W_{xo} y_{t-1} + W_{ho} h_{t-1} + b_o), \tag{2.12}$$

$$\tilde{c}_t = \tanh(W_{xc} y_{t-1} + W_{hc} h_{t-1} + b_c), \tag{2.13}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.14}$$

$$h_t = o_t \odot \tanh(c_t), \tag{2.15}$$

where all W and b are the weight matrices and bias vectors. σ is the sigmoid function and $\odot$ is the element-wise multiplication. i_t, f_t, o_t, $\tilde{c}_t$, c_t, and h_t are the input gate, forget gate, output gate, cell input, cell state, and hidden state at time step t, respectively. y_{t-1} is the input word at time step t. These computations can be visualized in Figure 2.1.

The intuition of the long short-term memory cells is to make the model itself choose to remember or forget. The forget gate f_t controls the amount of information to be forgotten from the previous cell state c_{t-1}. The input gate i_t controls the amount of information to be added to the cell state c_t. The output gate o_t controls the amount of information to be output from the cell state c_t.

Conditional language modeling

One of the tasks of interest in natural language processing is conditional language modeling. One example is the translation task, where the generated text is conditioned on the given source language [24]. Formally, conditional language models additionally consider a set of conditioning variables $\boldsymbol{x}$ in addition to the partial se-

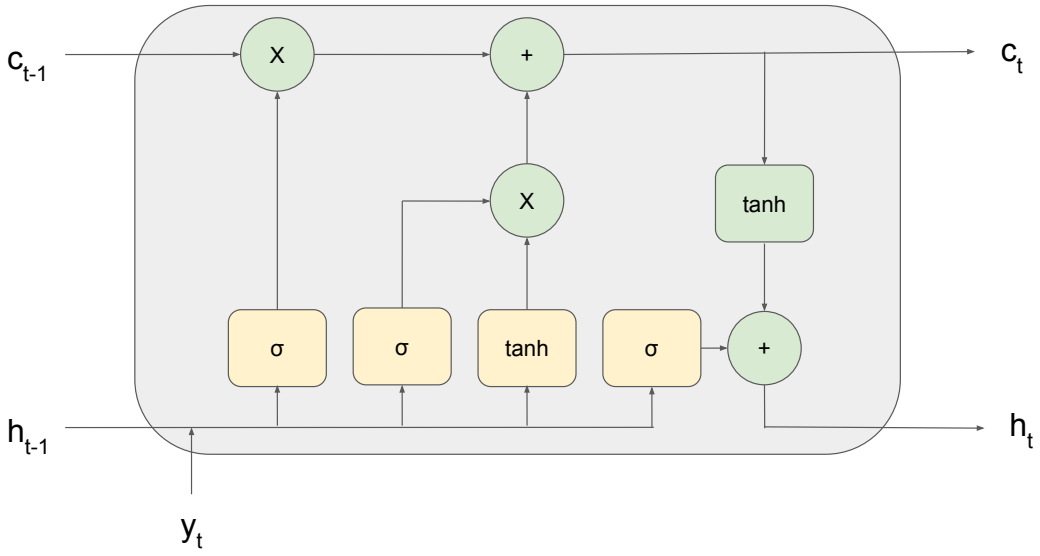

Figure 2.1: The long short-term memory cell. Every yellow cell indicates a neural network layer with the labeled activation function. The green cells are element-wise operations solely. The arrows indicate the flow of information.

quence $\boldsymbol{y}_{<t}$. The conditional language models are defined as:

$$P(\boldsymbol{y}|\boldsymbol{x}) \approx \prod_{t=1}^{T} \hat{P}_{\mathrm{LM}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}), \tag{2.16}$$

where $\boldsymbol{x}$ is the condition variable. W_o and b_o are the output weight matrix and bias vector.

This formulation comes in handy when the task naturally has a source sentence and a target sentence. Take machine translation as an example: the condition variable $\boldsymbol{x}$ can be the original language to be translated, and the target sentence $\boldsymbol{y}_{<t}$ is the already translated part.

The conditional language models are usually built with the encoder–decoder framework [6, 25]. The encoder is responsible for encoding the source sentence into a single vectorial representation. The decoder then decrypts the representation into the target

sentence. The general encoder–decoder framework works in the following way:

$$c = \text{Encoder}(\boldsymbol{x}), \tag{2.17}$$

$$h_t = f(h_{t-1}, y_{t-1}, c). \tag{2.18}$$

The Encoder function is usually a neural network. The vector c is the representation of the source sentence. The hidden state h_t is the hidden state of the decoder at time step t. The function f is the recurrent function that updates the hidden state h_t based on the previous hidden state h_{t-1}, the previous word y_{t-1}, and the source representation c.

The attention mechanism for conditional language modeling

Although the conditional language model with long short-term memory is able to capture the long-term dependency, they still suffer from the problem of information bottleneck caused by the fixed-length hidden state. The fixed-step hidden state is not able to capture all the information in the source sentence due to the compression loss from the sequence $\boldsymbol{x}$ to a single vector c. To address this issue, the attention mechanism is introduced to the recurrent neural network [24]. The attention mechanism is defined as:

$$s_t = \text{Encoder}(\boldsymbol{x}_{<t}), \tag{2.19}$$

$$e_{t,i} = a(h_{t-1}, s_i), \tag{2.20}$$

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{k=1}^{T} \exp(e_{t,k})}, \tag{2.21}$$

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} s_i, \tag{2.22}$$

$$h_t = f(h_{t-1}, y_{t-1}, c_t). \tag{2.23}$$

As shown in the above equations, the attention mechanism computes a set of attention weights $\alpha_{t,i}$ for each word s_i in the source sentence. The attention weights are computed based on the previous hidden state h_{t-1} and the source representation

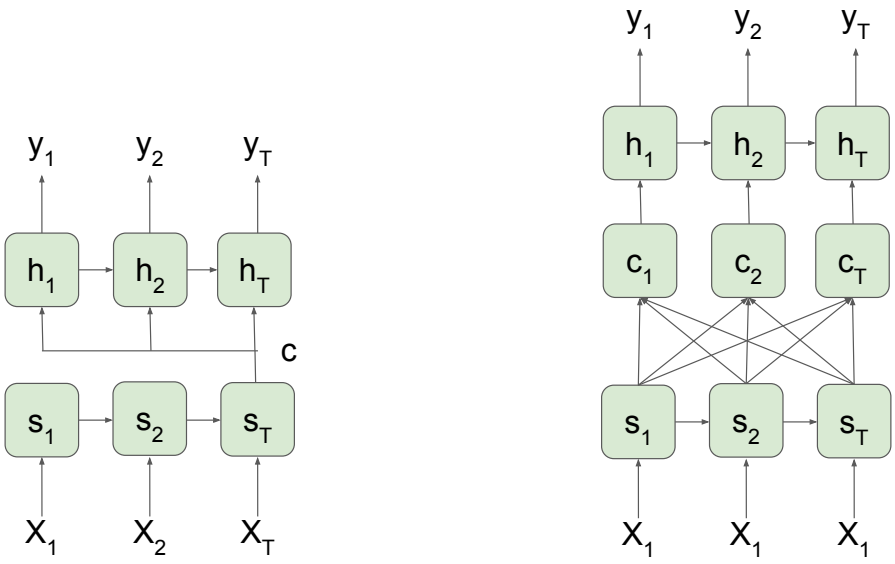

Figure 2.2: The encoder–decoder architecture without (left) and with (right) the attention mechanism.

s_i. The attention weights are then used to compute the context vector c_t. The context vector c_t is a weighted sum of the source representation s_i. The context vector c_t is then used to update the hidden state h_t. This practice significantly reduces the information bottleneck problem.

In Figure 2.2, we visualize the difference of encoder–decoder networks with and without the attention mechanism. The left part of Figure 2.2 shows the encoder–decoder network without the attention mechanism. In this case, the decoder only has access to the last hidden state of the encoder. The single vector c is then used by every time step of the decoder. The right part of Figure 2.2 shows the encoder–decoder network with the attention mechanism. In this case, the attention mechanism generates a sequence of context vectors c_t for each time step of the decoder. This enables the decoder to decide which source words to focus on at each time step. This practice is hence named "attention".

The vanilla Transformer model for language modeling

The long short-term memory with the attention mechanism can capture the information of the previous words more effectively. However, a key limitation of the recurrent neural network is the sequential computation. In other words, for each step t, the model needs to use the previously generated h_{t-1} to compute h_t. This recurrence is undesired for modern highly parallelized training. To address this issue, the Transformer model is introduced [1] to replace the recurrent neural network with the attention mechanism.

The vanilla Transformer model is composed of two parts: the encoder and the decoder. The encoder is a stack of L identical encoder layers, while the decoder is a stack of L decoder layers. For each encoder layer, there are two sub-layers: the multi-head self-attention layer and the feed-forward layer. The multi-head self-attention layer is used to capture the long-term dependency.

The decoder layer is similar to the encoder layer. The major architectural difference is that the decoder layer has an additional multi-head attention layer, which is used to capture the dependency between the source sentence and the target sentence. In addition, the attention pattern in this layer is masked to prevent the decoder from attending to future words.

We will present the details of the multi-head self-attention layer and the multi-head attention layer in the next chapter.

Encoder–decoder and decoder-only Transformer models

Although the original Transformer was proposed to be an encoder–decoder model, a recent trend is to use the decoder-only Transformer model for language modeling. A representative is the GPT family [5, 26]. The decoder-only Transformer model is able to achieve comparable or even better performance than the encoder–decoder Transformer model.

It is worth noting that the decoder-only Transformer model can be used for both

conditional and non-conditional language modeling. It is straightforward to see the application of the non-conditional case, where the decoder-only Transformer model is used to generate the next word given the previous words. For the conditional case, the decoder-only Transformer model can be used to generate the next word given the previous words and the source sentence. The source sentence and target sentences are concatenated as the input to the decoder-only Transformer model. They are usually separated by a special token. The decoder-only Transformer model is able to capture the dependency between the source sentence and the target sentence, just as the encoder–decoder Transformer model does. More surprisingly, even when the model is solely trained on the non-conditional datasets, it turns out that the decoder-only Transformer model can generate the desired target sentence by "prompting". For example, let the task be English–German translation. If one wants to translate "I like to eat apples" to German, then simply showing a prompt "The translation of 'I like to eat apples' is" to the model may be enough for the model to generate "Ich esse gerne Äpfel" by completing the prompt.

Due to the success of the decoder-only Transformer model on both conditional and non-conditional tasks, we will not discriminate between the two types of tasks and focus on the non-conditional formulation in this thesis for the simplicity of our theory. It is easy to extend our theory to the conditional case.

2.2 Reinforcement learning

We include the background of reinforcement learning because it is a powerful tool for solving sequential decision-making problems. In this section, we will introduce the basic concepts of reinforcement learning and the policy gradient method. We will finish this section by explaining how the policy gradient methods can be applied to the language modeling problem.

2.2.1 Markov decision process

The reinforcement learning problem is formulated as a Markov decision process. An episodic Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition probability, $\mathcal{R}$ is the reward function, and H is the horizon length. At each time step t, the agent observes the state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$. The agent then receives a reward $r_t \in \mathcal{R}$ and transits to the next state $s_{t+1} \in \mathcal{S}$ according to the transition probability $\mathcal{T}(s_{t+1}|s_t, a_t)$. The goal of the agent is to maximize the expected cumulative reward $\mathbb{E}[\sum_{t=0}^{H} r_t]$. The horizon length H is used to determine the maximum length of the episode.

2.2.2 Policy

The policy π is a function that maps the state s_t to the action a_t. The policy can be either deterministic or stochastic. The deterministic policy is defined as $\pi : \mathcal{S} \to \mathcal{A}$, whereas the stochastic policy is defined as $\pi : \mathcal{S} \times \mathcal{A} \to \Delta^{|\mathcal{A}|-1}$. The stochastic policy is usually parameterized by a neural network, which takes the state s_t as the input and outputs a probability distribution over the action space $\mathcal{A}$. The action is then sampled from the probability distribution.

With the policy, the agent can take actions in the environment. This is called a *rollout* process used to collect trajectories. A trajectory is a sequence of state–action pairs $(s_0, a_0, r_0), (s_1, a_1, r_0), \ldots, (s_H, a_H, r_H)$. The rollout starts from the initial state s_0 and ends at the terminal state s_H. The trajectory is collected following the procedure below:

$$s_0 \sim \mathcal{S}_0, \tag{2.24}$$

$$a_t \sim \pi(s_t), \tag{2.25}$$

$$s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a_t), \tag{2.26}$$

$$r_t \sim \mathcal{R}(r_t|s_t, a_t). \tag{2.27}$$

This procedure is illustrated in Figure 2.3. The gray cell is the environment that

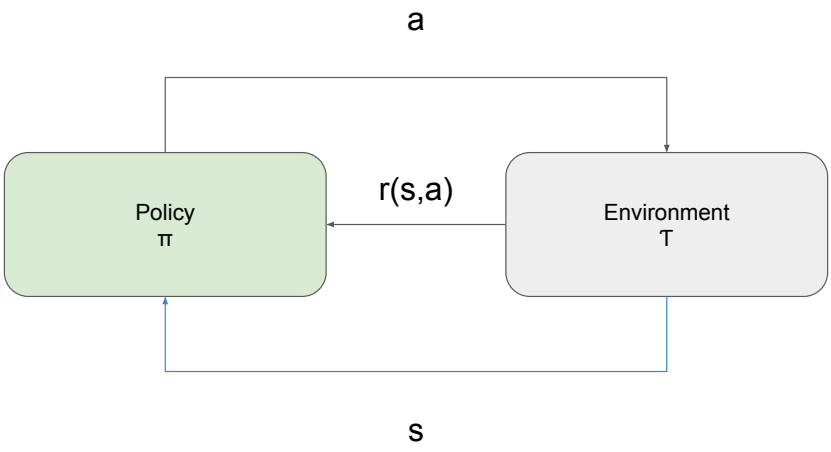

Figure 2.3: The rollout process. Adapted from [29].

the learning agent could not control. Instead, the agent learns to make optimal decisions by interacting with the environment. This is achieved by finding a good policy π, represented by the green cell. In this figure, we omit the implementation of the policy. In practice, the implementation is quite flexible. The simple solution could directly use a neural network to generate an action given the state as the input. The policy can also be implemented with a value function and a policy function, which is called the actor-critic method [27]. It is worth noting that the policy can be implemented without actually modeling a probability over actions. One example is the deep Q-learning [28] algorithm, where the actions are chosen greedily with respect to the action-value function.

2.2.3 Value equations and Bellman equation

The value function is used to evaluate the goodness of a state or a state–action pair. The state-value function $v^{\pi}(s_t)$ is defined as the expected cumulative reward starting

from the state s_t and following the policy π:

$$v^{\pi}(s_t) = \mathbb{E}_{\pi}\Big[\sum_{i=t}^{H} r_i \Big| s_t\Big]. \tag{2.28}$$

In addition to the state-value function, the action-value function is also defined as the expected cumulative reward starting from the state s_t, taking any action a_t and following the policy π afterward:

$$q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\Big[\sum_{i=t}^{H} r_i \Big| s_t, a_t\Big]. \tag{2.29}$$

The value functions can be computed recursively by the Bellman equation:

$$v^{\pi}(s_t) = \mathbb{E}_{\pi}\Big[r_t + \sum_{i=t+1}^{H} r_i \Big| s_t\Big] \tag{2.30}$$

$$= \mathbb{E}_{\pi}\Big[r_t + v^{\pi}(s_{t+1}) \Big| s_t\Big]. \tag{2.31}$$

For the action-value function, we also have:

$$q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\Big[r_t + \sum_{i=t+1}^{H} r_i \Big| s_t, a_t\Big] \tag{2.32}$$

$$= \mathbb{E}_{\pi}\Big[r_t + q^{\pi}(s_{t+1}, a_{t+1}) \Big| s_t, a_t\Big]. \tag{2.33}$$

One interesting property of the value functions is that they satisfy the following equations:

$$v^{\pi}(s_t) = \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) q^{\pi}(s_t, a_t), \tag{2.34}$$

$$q^{\pi}(s_t, a_t) = r_t + \sum_{s_{t+1} \in \mathcal{S}} \mathcal{T}(s_{t+1}|s_t, a_t) v^{\pi}(s_{t+1}). \tag{2.35}$$

These equations provide the critical property of the value functions. By constructing the value functions, we can evaluate the goodness of any given policy π in a principled way.

2.2.4 Policy gradient

The goal of the agent is to find the optimal policy π^* that maximizes the expected cumulative reward $\mathbb{E}[\sum_{t=0}^{H} r_t]$. The policy gradient is a class of methods that directly optimize the policy π to maximize the expected cumulative reward. It is based on the following equation:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{H} \nabla_\theta \log \pi(a_t|s_t)(q^\pi(s_t, a_t) - b(s_t)) \right], \tag{2.36}$$

where $b(s_t)$ is the baseline function only depending on the state s_t.

The simplest policy gradient method is the REINFORCE algorithm [30], where the baseline function is set to zero and $q^\pi(s_t, a_t)$ is replaced by the empirical return $G_t = \sum_{i=t}^{H} r_i$ from a single rollout. Unlike other advanced algorithms like actor–critic methods [27] or proximal policy optimization [31], this implementation does not require the value function to be computed. The only trainable parameters are the parameters of the policy π.

2.3 Text generation

Besides training a language modeling using the teacher forcing objective that teaches the model to predict the next word given the previous words, there are many other methods for text generation. In this section, we will introduce several categories of text generation methods beyond language modeling trained with teacher forcing.

2.3.1 Text generation using reinforcement learning

Interestingly, the language modeling problem can be easily formulated as a reinforcement learning problem. We can design the state space $\mathcal{S}$ as the history of the tokens generated so far. The action space $\mathcal{A}$ is the vocabulary of the language. The transition function $\mathcal{T}$ is a function that concatenates the new token to its context. The language model can then serve as a policy π that maps the state s_t to the probability

distribution over the action space $\mathcal{A}$.

This formulation has been extensively used in text generation. For instance, Sokolov *et al.* [32] and Kreutzer *et al.* [33] leverage the bandit-structured prediction framework for text generation with BLEU as the heuristically defined reward. Bahdanau *et al.* [34] and Shen *et al.* [35] utilize different variants of policy gradient for RL training. However, these methods are task-specific and suffer from the problem of sparse rewards where most of the feedback is zero. More importantly, these approaches require labeled data to calculate the reward and cannot utilize unlabeled data. To address this, Wu *et al.* [36] propose to learn a reward regression model on the labeled dataset and perform RL on the unlabeled data with the learned reward. As mentioned, such a method is still task-specific because it requires the human heuristics of the task to define the proper reward function.

To summarize, reinforcement learning is a promising approach for text generation by treating the language model as a policy. However, the reward function is a critical component that is missing. In addition, they can hardly be applied to unlabeled data because they require groundtruth text to compute the reward. These drawbacks prevent the reinforcement learning methods from being widely used in text generation.

2.3.2 Text generation with unlabeled data

There are vastly available unlabeled data (e.g., the crawled web text) unused in conditional language modeling, as it requires human annotations for source–target pairs. In this section, we will introduce several methods that can utilize unlabeled data for text generation.

Semi-supervised training. A popular paradigm for using unlabeled data is semi-supervised learning, which uses the unlabeled data as a supplement to enhance performance based on the labeled data solely. Two popular semi-supervised learning methods are self-training [37, 38] and back-translation [14]. Both methods first train

a model on the labeled data and then generate pseudo-labeled pairs for the unlabeled sentences. The difference is that self-training generates pseudo-labeled pairs from source to target, whereas back-translation generates from target to source.

However, such methods cannot efficiently utilize the data because of the lack of exploration. Specifically, they do not let the model generate possible sentences during training. Additionally, the exposure bias issue remains because they are trained with the teacher-forcing objective.

Unsupervised training. There exists some work that trains the model on unlabeled data solely. For example, Lample *et al.* [39] propose a framework that trains a machine translation model using the monolingual data only. However, such methods are strongly task-specific and do not generalize well. Most importantly, they are not able to achieve comparable results to the supervised or semi-supervised methods. Given the inferior performance, we do not consider them in this thesis.

2.3.3 Other text generation methods

A line of work uses the *generative adversarial network* (GAN) [40] to alleviate the issue of exposure bias. For example, Yu *et al.* [41] and Guo *et al.* [42] propose to use GAN-style training to generate text similar to the training set in an on-the-fly manner. This practice reduces the discrepancy between training and inference because GAN sends its own generation as inputs rather than using groundtruth sentences during training. Shi *et al.* [43] further formulate the adversarial training using the IRL interpretation. These GAN-style methods are different from ours in two main ways. First, GAN-style training requires labeled corpora and thus cannot be directly applied to semi-supervised learning on unlabeled datasets. Second, GAN-style training involves the optimization of an adversarial objective, making the training unstable, e.g., suffering from mode collapse [40].

Search is also a popular way to replace teacher forcing. The Learning to Search

(L2S) framework [44, 45] enables the model to search for a better score during learning and is widely applied to text generation. For example, Wiseman and Rush [46] propose to optimize the beam search results through training. Li *et al.* [47] develop an unsupervised learning approach to text generation based on local search. In addition to the L2S framework, Leblond *et al.* [48] leverage the Monte Carlo tree search [49, 50] to select better tokens in a step from the sampled generation. These methods are different from ours since they need either heuristically defined scoring functions or labeled data, limiting their methods to certain tasks or to the supervised paradigm. However, given the success of these methods, we consider the search-based approach an interesting future extension of our work.

The intuition behind our work is also related to *imitation learning* methods in general. Typically, these methods aim to obtain a good policy given a dataset containing state–action pairs. The easiest approach is behavior cloning [51], which greedily imitates the demonstration. Similar to the exposure bias, behavior cloning also faces the problem of compounding errors [52]. SMILe [52] and DAgger [53] mitigate the problem by querying an expert. In text generation, Du and Ji [54] empirically verify that imitation learning methods are helpful. Recently, Pang and He [55] frame the text generation task as an offline reinforcement learning problem, which learns from a dataset containing tuples of state, action, and reward. Compared with our method, these approaches rely on labeled sentence pairs and cannot effectively make use of unlabeled datasets.

Chapter 3

Language models learn reward functions implicitly

3.1 Overview

In this chapter, we build a simple connection between the teacher forcing objective and reward learning for text generation. We first introduce the Transformer architecture, which is the foundation of our approach. We then formally present how to formulate text generation as a Markov decision process. Finally, we derive a reward function from a language model trained by teacher forcing.

3.2 Neural architecture

As discussed in Section 2.1, we use the Transformer architecture to model the conditional probability of a sequence $\boldsymbol{y} = (y_1, \cdots, y_T)$ given a source sequence $\boldsymbol{x} = (x_1, \cdots, x_S)$. Specifically, we choose a well-known pre-trained model, T5 [56], as our base model. T5 is a Transformer encoder–decoder model with a vocabulary size of 32,000. We use the pre-trained T5 model as initialization for all settings.

The critical component of the Transformer architecture is the attention mechanism. Different from the recurrent neural network-based models introduced in Section 2.1.3, the attention mechanism in Transformer allows parallel computation within every attention layer. For each layer, the attention mechanism consists of three parts: the query Q, the key K, and the value V. The attention mechanism computes the

attention weights between the query and the key, and then the weighted sum of the value is the output of the attention layer. The attention weights are computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3.1}$$

where d_k is the dimension of the key.

The attention mechanism in Transformer is further improved by the multi-head design, which computes multiple attention patterns in parallel. The output is the concatenation of the result from each head, formally defined as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O, \tag{3.2}$$

where each head is defined as follows:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{3.3}$$

The multi-head attention is important to the Transformer architecture. It allows the model to capture different attention patterns in different representation subspaces and recombine them together. The multi-head attention is used in both the encoder and the decoder of the Transformer architecture. However, they are used in different ways.

In the encoder, the multi-head attention is used to capture the relationship between the source sequence itself, which is known as the self-attention mechanism.

In the decoder, there are two multi-head attention layers. The first one is the self-attention layer, similar to which in the encoder. One key difference is that it requires the causal mask on the attention patterns to prevent the model from seeing the future tokens. This is because the decoder is auto-regressive and can only see the previous tokens during inference. The second one is the encoder–decoder attention layer, used to capture the relationship between the source sequence and the target sequence. The query is generated from the previous self-attention layer in the decoder, and the key and the value are from the last encoder layer.

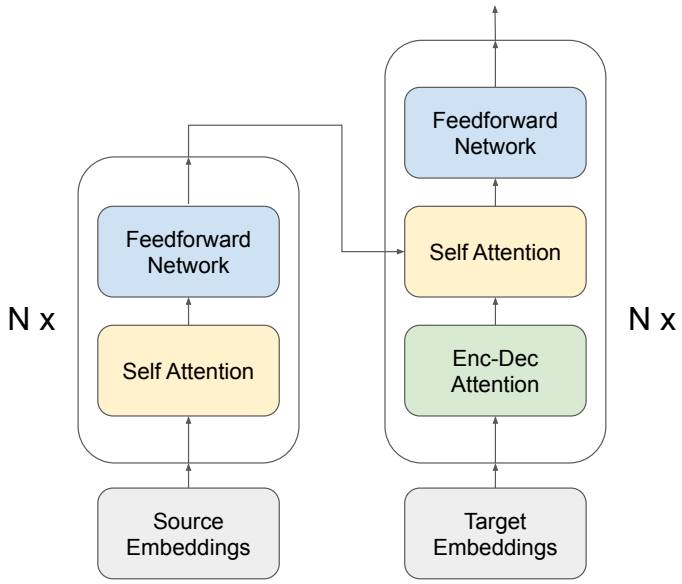

Figure 3.1: The Transformer architecture. The layer normalizations, activation functions, and residual connections are omitted for simplicity. Adapted from [1].

Besides the multi-head attention layers, the Transformer encoder and decoder also contain feed-forward layers. The feed-forward layer is a two-layer fully-connected network with a ReLU activation function in between. The feed-forward layer is used to transform the representation of the multi-head attention layer. The feed-forward layer is used in both the encoder and the decoder. In addition, layer normalization and residual connection are applied to coordinate these layers.

In summary, the Transformer architecture is illustrated in Figure 3.1. The yellow cells are self-attention layers, where they only attend to the sentences within the encoder or the decoder. The green cells are encoder–decoder attention layers, where they attend to the outputs of the encoder modify the representation of the decoder. The blue cells are feed-forward layers. All layer normalizations, activation functions, and residual connections are omitted for simplicity.

3.3 Model training and inference

In this section, we will introduce the typical training procedure for the Transformer architecture. The training is usually called teacher forcing. The teacher forcing training is a supervised learning method. It requires the groundtruth $\boldsymbol{y}$ of the target sequence corresponding to the source sentence $\boldsymbol{x}$. The model is trained to maximize the log-likelihood of the groundtruth sequence. The log-likelihood for each sentence is defined as follows:

$$\max_{\theta}\{-L_{\mathrm{TF}}(\theta;\boldsymbol{x},\boldsymbol{y}) = \log P_\theta(\boldsymbol{y}|\boldsymbol{x}) = \sum_{t=1}^{T} \log P_\theta(y_t|\boldsymbol{y}_{<t},\boldsymbol{x})\}, \tag{3.4}$$

where θ is the parameters of the transformer model.

During inference, the model is used to generate the target sequence token by token. The model generates the next token based on the previous tokens. There are different ways to generate the next token. The most common way is to use the greedy search. The greedy search chooses the token with the highest probability as the next token, which is expressed as follows:

$$y_t = \arg\max_{y\in\mathcal{V}} P_\theta(y|\boldsymbol{y}_{<t},\boldsymbol{x}). \tag{3.5}$$

After the next token is generated, it is concatenated with the previous tokens $\boldsymbol{y}_{<t}$ to be $\boldsymbol{y}_{<t+1}$. The process is repeated until the end-of-sentence token is generated.

As shown in Chapter 1, this long-standing practice has been shown to be problematic. It usually requires reinforcement learning to improve the performance of the model. However, given the reward functions are not easy to design, the reinforcement learning methods are not widely used in the text generation tasks. In the next chapter, we will introduce our method to automatically learn a proper reward model for language models.

3.4 Our method and procedure

Our approach trains the language model on non-parallel data with reinforcement learning, whose foundation is the Markov decision process. In this section, we first introduce the Markov decision process formulation for text generation. Then we describe our method to derive the reward function from a language model trained by teacher forcing. Finally, we describe the policy gradient method used in RL training with our induced reward function.

3.4.1 RL formulation of text generation

Text generation as a Markov decision process (MDP). We formulate the text generation process as an (undiscounted) MDP, which can be represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$. At every step, a decision $a \in \mathcal{A}$ is made based on its state $s \in \mathcal{S}$. The transition dynamic $\mathcal{T}(s'|s, a)$ is the probability of the next state being s', given the current state s and the action a. A function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines the reward based on a state and an action.

Typically, the decision making is assisted by a policy π, which is a predicted distribution over actions and is trained to maximize the expected total reward, also known as an action value function:

$$q^{\pi}(s, a) := \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=1}^{H} r(s_t, a_t) | s_1 = s, a_1 = a \right], \tag{3.6}$$

where H is the number of steps. Theoretical results show that the optimal policy π^* satisfies the Bellman optimality equation:

$$q^{\pi^*}(s, a) = r(s, a) + \sum_{s' \in \mathcal{S}} T(s'|s, a) \max_{a'} q^{\pi^*}(s', a'). \tag{3.7}$$

For text generation, the MDP state can be defined as the partial generated sequence $\boldsymbol{y}_{<t} := (y_1, \cdots, y_{t-1})$, and the action as the next token y_t in the vocabulary $\mathcal{V}$. The transition dynamic $T(\cdot|s, a)$ here is deterministic, since every state–action pair $(\boldsymbol{y}_{<t}, y_t)$ leads to a unique state $\boldsymbol{y}_{<t+1}$ for the next step.

In previous RL-based text generation, there lacks a naturally defined reward function $r(s, a)$. While researchers have applied various heuristics as the reward [34, 35], they suffer from several shortcomings (e.g., sparsity and task specificity) as mentioned in Chapter 1. To address these problems, we propose to induce a reward function for text generation tasks in a principled approach by inverse reinforcement learning.

Inverse Reinforcement Learning (IRL). The goal of IRL is to learn a reward function $r(s, a)$. Especially, we wish the resulting action value function q computed by Eqn. (3.6) could satisfy $q(s, a) \geq q(s, a')$ for every $a' \in \mathcal{A}$ and every (s, a) pair in the training set $\mathcal{D}$. In other words, the decisions in $\mathcal{D}$ are made greedily by $\text{argmax}_a\, q(s, a)$ given any state s. Unfortunately, Ng and Russell [57] show that this is an ill-posed problem since the desirable reward function r is not unique. Therefore, we follow a common assumption [58–60] to resolve the ambiguity:

Assumption 1 *Given an action value function q, the policy π takes the form of* $\pi_q(a|s) := \exp(q(s, a))/\sum_{a'} \exp(q(s, a'))$.

In traditional IRL [58–60], reward learning is difficult and this assumption does not directly yield a reward function due to the stochastic state transition $T(s'|s, a)$. However, our insight is that the transition is deterministic for text generation tasks, and thus we may utilize Assumption 1 to induce an action value function q, and then a reward function r, from some learned policy π, as explained in the next part.

3.4.2 Teacher forcing recovers IRL

One of our main contributions is that we show the seemingly complicated reward learning in Section 3.4.1 can be recovered by teacher forcing, the *de facto* common practice of supervised text generation. Our discovery leads to a convenient approach that derives a step-wise reward function simply from general language models, without the need for task-specific heuristics. This makes RL more general for text generation, and our step-wise reward largely simplifies RL training.

Maximum likelihood estimation (MLE) for IRL. Following Assumption 1, we let the policy $\pi_{q_\omega}(\cdot|s) \propto \exp(q_\omega(s,\cdot))$, where q_ω is a parameterized action value function. Under such a policy, the probability of each trajectory $\tau := ((s_1, a_1), \ldots, (s_{|\tau|}, a_{|\tau|}))$ in the dataset is given by the trajectory distribution $P^{\pi_{q_\omega}}$. The likelihood of the dataset is given by

$$P_{\mathrm{IRL}}(\mathcal{D}|\omega) := \prod_{\tau\in\mathcal{D}} P^{\pi_{q_\omega}}(\tau). \tag{3.8}$$

Teacher-forcing training. For text generation, the standard teacher-forcing training is to minimize the loss:

$$L_{\mathrm{TF}}(\omega;\mathcal{D}) := -\sum_{\boldsymbol{y}\in\mathcal{D}}\sum_{t=1}^{|\boldsymbol{y}|} \log p_\omega(y_t|\boldsymbol{y}_{<t}), \tag{3.9}$$

where the predicted probability of the next token being v is $p_\omega(v|\boldsymbol{y}_{<t}) = \frac{\exp(f_\omega(\boldsymbol{y}_{<t},v))}{\sum_{v'\in\mathcal{V}}\exp(f_\omega(\boldsymbol{y}_{<t},v'))}$ for the logit function f_ω with parameters ω. In language model training, an additional input $\boldsymbol{x}$ may be added to the conditional probabilities but is omitted here for simplicity.

The below theorem shows their equivalence up to an additional constant.

Theorem 2 *Suppose the value function q in Eqn.* (3.8) *and the language model f in Eqn.* (3.9) *have the same parametrization ω, we have*

$$L_{\mathrm{TF}}(\omega; D) = -\log P_{\mathrm{IRL}}(\mathcal{D}|\omega) + \mathrm{const}\,. \tag{3.10}$$

Proof.

For the MLE of IRL under Assumption 1, the Ionescu–Tulcea theorem [61] asserts that there exists a unique trajectory distribution P^π_μ satisfying

$$\begin{aligned}
&P^\pi_\mu(s_1) = \mu(s_1),\\
&P^\pi_\mu(s_1, a_1, \ldots, s_t, a_t) = P^\pi_\mu(s_1, a_1, \ldots, s_t)\pi(a_t|s_t),\\
&P^\pi_\mu(s_1, a_1, \ldots, s_t, a_t, s_{t+1}) = P^\pi_\mu(s_1, a_1, \ldots, s_t, a_t)T(s_{t+1}|s_t, a_t)
\end{aligned}$$

for any $t \geq 1$, given the initial state distribution μ, transition probability T, and policy π.

The likelihood can thus be factorized by the multiplication of μ, T, and π:

$$P_{\text{IRL}}(\mathcal{D}|\omega) = \prod_{\tau \in \mathcal{D}} P_{\mu}^{\pi_{q_\omega}}(\tau) = \prod_{\tau \in \mathcal{D}} \left[\mu(s_1) \pi_{q_\omega}(a_1|s_1) \prod_{t=2}^{|\tau|} T(s_t|s_{t-1}, a_{t-1}) \pi_{q_\omega}(a_t|s_t) \right].$$

As mentioned, text generation has a deterministic transition, i.e., $T(s'|s,a) = 1$ for the next state $s' = s + [a]$. Taking the μ terms out, we have

$$-\log P_{\text{IRL}}(\mathcal{D}|\omega) = -\log \prod_{\tau \in \mathcal{D}} \prod_{t=1}^{|\tau|} \pi_{q_\omega}(a_t|s_t) - \log \prod_{\tau \in \mathcal{D}} \mu(s_1), \tag{3.11}$$

where the second term is a constant in terms of ω. In Section 3.4.1, text generation is modeled as an MDP with $s_t = \boldsymbol{y}_{<t}$ and $a_t = y_t$. Therefore, the first term of Eqn. (3.11) is the same as Eqn. (3.9) under the parametrization $\pi_{q_\omega} = p_\omega$, concluding the equivalence between MLE for IRL and the teacher-forcing training of a language model. ■

Inducing the reward function. Theorem 2 shows that language model training with teacher forcing actually learns an IRL model. Thus, we may derive a reward function assuming the action value function is well trained:

$$r(s,a) = q_\omega(s,a) - \sum_{s' \in \mathcal{S}} T(s'|s,a) \max_{a' \in \mathcal{A}} q_\omega(s', a') = f_\omega(s,a) - \max_{a' \in \mathcal{A}} f_\omega(s + [a], a'), \tag{3.12}$$

where the first equality is due to the Bellman optimality condition (3.7); the second equality is due to the parametrization of $q_\omega = f_\omega$ and the deterministic transition $T(s'|s,a) = 1$ for s' being the concatenation of the prefix s and token a.

Remark 3 *It is easy to notice that $f_\omega(s, \cdot)$ may be arbitrarily shifted by a constant c_s without changing π_ω. This also shifts the derived reward $r(s,a)$ by $c_s - c_{s+[a]}$. However, it does not affect the optimal policy. We will prove this in Theorem 5 after introducing policy gradient methods.*

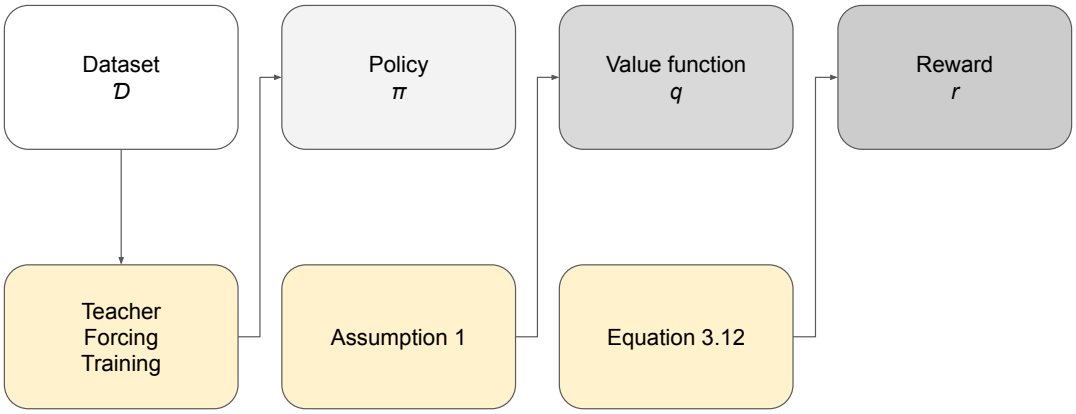

Figure 3.2: The process of deriving the reward function.

Our use of the Bellman optimality condition is different from classic RL, where the reward is well-defined and the action value function is thus learned [29]. Instead, we induce the underlying reward assuming the action value function is known (given by Assumption 1). Figure 3.2 shows the whole process of our derivation.

In real-world applications, the learned action value function might be imperfect; in this case, we may bound the error of our induced reward with the following theorem.

Theorem 4 *Let r^* be an underlying true reward function and q^* be the corresponding optimal value function. Given an approximate value function q, we denote by r the reward function derived from Eqn.* (3.12)*. Then, we must have $\|r - r^*\|_\infty$ bounded by $O(\|q - q^*\|_\infty)$. Here, $\|\cdot\|_\infty$ takes the maximum absolute value over all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.*

Proof. For any $s \in \mathcal{S}$ and $a \in \mathcal{A}$, we have

$$
\begin{aligned}
&|r(s,a) - r^*(s,a)| \\
=&|q(s,a) - q^*(s,a) + \max_{a'} q^*(s+[a],a') - \max_{a'} q(s+[a],a')| && (3.13)\\
\leq&|q(s,a) - q^*(s,a)| + |\max_{a'} q^*(s+[a],a') - \max_{a'} q(s+[a],a')| && (3.14)\\
\leq&\max_{s',a'} |q(s',a') - q^*(s',a')| + \max_{s'} |\max_{a'} q^*(s',a') - \max_{a'} q(s',a')| && (3.15)\\
\leq&\max_{s',a'} |q(s',a') - q^*(s',a')| \\
&+ \max_{s'} \max\{\max_{a'} q^*(s',a') - \max_{a'} q(s',a')), \max_{a'} q(s',a') - \max_{a'} q^*(s',a))\} && (3.16)\\
\leq&\max_{s',a'} |q(s',a') - q^*(s',a')| \\
&+ \max_{s'} \max\{\max_{a'}(q^*(s',a') - q(s',a')), \max_{a'}(q(s',a') - q^*(s',a'))\} && (3.17)\\
\leq&2\max_{s',a'} |q(s',a') - q^*(s',a')| && (3.18)\\
=&2\|q - q^*\|_\infty. && (3.19)
\end{aligned}
$$

Here, Eqn. (3.13) is from the Bellman equation; Eqn. (3.14) follows the triangle inequality; and Eqn. (3.15) generalizes certain s and a to all possible $s' \in \mathcal{S}, a' \in \mathcal{A}$. Eqn. (3.16) discusses two possible cases: whether $\max_{a'} q(s',a') \geq \max_{a'} q^*(s',a')$ or not. Eqn. (3.17) is because $-\max_{a'} q(s',a') \leq -q(s',a'')$ for any $a'' \in \mathcal{A}$. Eqn. (3.18) merges all the maximum operation, and Eqn. (3.19) is the definition of the infinity norm.

Since the last equation does not depend on s and a, we conclude $\|r - r^*\|_\infty$ is bounded by $O(\|q - q^*\|_\infty)$. ■

3.4.3 Periodically synchronized behavior policy

In text generation, a neural network can be viewed as a policy π that predicts the word distribution given the state of a decoding step. The reward induced from Section 3.4.2 can be used to improve the policy through RL. To stabilize training, we propose a variant of off-policy policy gradient methods [62] with a periodically synchronized behavior policy.

Our RL training adopts the off-policy REINFORCE [30] as the backbone of our algorithm. Let π_φ be the model policy (i.e., the model's prediction) to be optimized, and π_b be the behavior policy (i.e., the sampling distribution during training). Through importance sampling, the gradient of the expected total reward with respect to φ can be obtained by the off-policy policy gradient theorem [62]

$$\nabla_\varphi \mathop{\mathbb{E}}_{\pi_\varphi}\left[\sum_t r(s_t, a_t)\right] = \mathop{\mathbb{E}}_{\pi_b}\left[\sum_t \rho_t \hat{q}_r(s_t, a_t) \nabla_\varphi \log \pi_\varphi(a_t|s_t)\right], \tag{3.20}$$

where $\rho_t := \pi_\varphi(a_t|s_t)/\pi_b(a_t|s_t)$ is the importance weight, and $\hat{q}_r(s_t, a_t) := \sum_{i \geq t} r(s_i, a_i)$ is the total reward of the trajectory. In practice, off-policy REINFORCE ($\pi_\varphi \neq \pi_b$) is more exploratory than the on-policy one ($\pi_\varphi = \pi_b$), since the model policy π_φ would become more concentrated during optimization and does not explore much, whereas π_b is typically chosen to cover more trajectories. However, Degris *et al.* [62] adopt a fixed behavior policy π_b, which does not perform exploitation according to the current model policy. The lack of exploitation might lead to less informative training.

To balance exploration and exploitation, we would like the behavior policy to be close to the model policy but stay exploratory at the same time. We thus propose a periodically updating schedule, where the behavior policy is frozen for a long period to encourage exploration but keeps track of the current model policy to enhance exploitation. Particularly, we synchronize the behavior policy with the model policy for every k gradient updates of the latter (e.g., $k = 5000$). Our remedy is a simple method overcoming the instability of REINFORCE. It shares a common ground with a number of policy gradient methods like the proximal policy optimization (PPO) [31], especially in that both methods involve multiple updates with a fixed behavior policy. As the main contribution of this paper is reward induction, we resort to this simple fix and leave the mathematical connection as an interesting future direction.

Algorithm 1 summarizes our approach. Our implementation is able to execute the loops in parallel, which speeds up the training process. Our periodically synchronized behavior policy further enables us to parallelize sampling and model updates to reduce

the awaiting time.

3.4.4 Application to semi-supervised learning

Our approach naturally aligns with the paradigm of semi-supervised learning, as it involves training a language model to induce the reward function, which requires (at least a small volume of) parallel data $\mathcal{D}_p$. Additionally, we assume there is a non-parallel dataset $\mathcal{D}_u$ containing input sentences only for RL training with the induced reward.

Our semi-supervised approach consists of two stages. We first train a language model f_ω on the parallel dataset $\mathcal{D}_p$ to induce the reward function r by Eqn. (3.12). The procedure is described in Section 3.4.2. The reward function then facilitates RL training on the non-parallel dataset $\mathcal{D}_u$, which is shown in Algorithm 1.

As mentioned in Remark 3, the reward $r(s, a)$ can be arbitrarily shifted by $c_s - c_{s+[a]}$. We show that this shift does not affect the optimal policy.

Theorem 5 *Suppose $r'(s, a) = r(s, a) + c_s - c_{s+[a]}$. Then the learned policies under $r'(s, a)$ and $r(s, a)$ are the same.*

Proof. By Eqn. (3.6), function q returns the expected total reward. In Algorithm 1, we sample it by

$$\begin{aligned}\hat{q}_t^{r'}(s_t, a_t) :=& r'(s_t, a_t) + r'(s_{t+1}, a_{t+1}) + \cdots + r'(s_{|\tau|}, a_{|\tau|}) \\ =& r(s_t, a_t) + c_{s_t} - \cancel{c_{s_{t+1}}} + r(s_{t+1}, a_{t+1}) + \cancel{c_{s_{t+1}}} - c_{s_{t+2}} + \cdots + r(s_{|\tau|}, a_{|\tau|}) + \cancel{c_{s_{|\tau|}}} \\ =& c_{s_t} + r(s_t, a_t) + r(s_{t+1}, a_{t+1}) + \cdots + r(s_{|\tau|}, a_{|\tau|}) =: \hat{q}^r(s_t, a_t) + c_{s_t}.\end{aligned}$$

The last line suggests the constant plays a role as the baseline in policy gradient, which is shown to be irrelevant to the optimal policy [29]. ■

3.5 Summary

We propose a novel approach to reward induction for RL training of conditional language modeling. Our approach is based on a common assumption in inverse re-

Algorithm 1: Our algorithm

Input: A non-parallel dataset $\mathcal{D}_u$, learned logit (value) function f_ω, policy π_φ with the initial parameter φ, total update steps U, and synchronizing period k

Output: A policy π_φ parameterized by φ

begin

  for $i \leftarrow 1...U$ **do**

    if $i \equiv 0 \pmod{k}$ **then**

      $\pi_b \leftarrow \pi_\varphi$; ▷ `Behavior policy update`

    Sample a source sentence $\boldsymbol{x} \in \mathcal{D}_u$

    Construct the initial state $s \leftarrow (\boldsymbol{x}, [\text{BOS}])$; ▷ `[BOS] is the beginning token`

    Sample a trajectory τ from the behavior policy π_b

    $\hat{q}^r_{h+1} \leftarrow 0$ and $g \leftarrow \mathbf{0}$

    for $t \leftarrow |\tau|...1$ **do**

      if $t = |\tau|$ **then**

        $r_t \leftarrow f_\omega(s_t, a_t)$; ▷ `Termination step, no` s_{t+1}

      else

        $r_t \leftarrow f_\omega(s_t, a_t) - \max_{a'} f_\omega(s_{t+1}, a')$; ▷ `By Eqn.` (3.12)

      $\hat{q}^r_t \leftarrow r_t + \hat{q}^r_{t+1}$; ▷ `Accumulating rewards`

      $\rho_t \leftarrow \pi_\varphi(a_t|s_t)/\pi_b(a_t|s_t)$; ▷ `Importance weight`

      $g \leftarrow g + \rho_t \hat{q}^r_t \nabla_\varphi \log \pi_\varphi(a_t|s_t)$; ▷ `By Eqn.` (3.20)

    $\varphi \leftarrow \varphi + \eta g$; ▷ `Gradient ascent`

  return π_φ

inforcement learning. We provide a principled way to induce the reward function from a learned logit function. We further propose a simple yet effective remedy to the instability of REINFORCE. Finally, we integrate our approach into the paradigm of semi-supervised learning. The experiment results will be presented in the next chapter.

Chapter 4

Results and discussion

4.1 Datasets description

Dialogue generation. We adopt two widely used datasets, DailyDialog [63] and OpenSubtitles [64], for the dialogue experiment. The DailyDialog dataset is constructed from English dialogues crawled from the Internet, whereas the OpenSubtitles dataset is constructed from movie subtitles based on IMDB identifiers. A dialogue session is split into single-turn context–response pairs in our experiment. For semi-supervised learning, we use the smaller dataset, DailyDialog, as the parallel corpus $\mathcal{D}_p$, and the larger dataset, OpenSubtitles, as the non-parallel corpus $\mathcal{D}_u$ (i.e., we only retain the context sentence in the OpenSubtitles dataset). This follows the common setup for semi-supervised learning, where the unlabeled dataset is larger than the labeled one.

It should be emphasized that a recent study [65] shows more than 20% of test samples are identical to some training samples in both DailyDialog and OpenSubtitles. This results in meaningless comparison and inflated performance of previous methods, e.g., a BLEU4 of 11.01 in AdaLabel [66] and 14.61 in DialogBERT [67]. Therefore, we use the deduplicated datasets in [65], containing 60K/6.5K/7K samples for training/validation/test in DailyDialog and 1M non-parallel samples in OpenSubtitles. Although our scores will be lower than previous inflated ones, we follow the correct setting for research.

We use BLEU scores [15] as main evaluation metrics, which are widely used in dialogue generation [65]. Following previous work [65], we lowercase all sentences and tokenize them with the NLTK library [68].

Paraphrase generation. We follow previous studies [69–71] and use the Quora Question Pair dataset[1] for the paraphrasing experiment. The Quora dataset is originally designed for paraphrase classification, containing both paraphrase and non-paraphrase pairs. The paraphrase pairs naturally form a parallel dataset for the generation purpose; following the common practice [71], we split it into 124K/4K/20K samples for training/validation/test. The non-paraphrase pairs, containing 510K sentences, are discarded in previous work, but we are able to utilize them in a semi-supervised manner.

We use the standard iBLEU score [72] as the main evaluation metric with an involvement af Self-BLEU (SBLEU) penalty, as the paraphrasing task requires using different lexicons. As introduced, the iBLEU score is calculated as $(1-\alpha)$ BLEU $-\alpha$ SBLEU. We follow the previous work and set α to 0.1 [69–71]. For clarity, we also report BLEU and S-BLEU scores in our experiment.

4.1.1 Settings and competing methods

For each task, we first fine-tune a T5-Base model [56] on the parallel data by Eqn. (3.9). Then we apply our proposed method to induce the reward and further train the model by Algorithm 1 on the non-parallel data. We compare our approach with the following semi-supervised methods.

Self-Training. We apply the supervised model to the non-parallel dataset and generate pseudo-target sentences, which are used to continue training the model. This is a commonly used semi-supervised approach in text generation literature [37, 38].

R-Regression. Wu *et al.* [36] propose a reward regression (R-Regression) ap-

[1] https://www.kaggle.com/c/quora-question-pairs

proach, where the reward is defined as the BLEU score. Since their reward is the same as the evaluation metric, such a method may achieve higher BLEU scores without actually improving the generation quality. By contrast, our reward is induced in a principled way and is agnostic to evaluation metrics. In our experiment, we replicate the R-regression method, which constitutes a controlled comparison to our approach, as the only difference is the reward function.

4.1.2 Implementation details

For all experiments, we initialize the model with T5-Base [56] provided by HuggingFace [73]. We use the label smoothing [74] with a coefficient of 0.1. We use the Adam [75] optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$. Each batch contains around 32K tokens.

For all conventional language model training, the learning rate is scheduled according to the original Transformer [1] with the warm-up steps set as 4000. For all RL training, we drop the warm-up phase and set the maximum learning rate to $1e-5$. We set the synchronizing period k to 5000. The reward of our method is scaled down by 100 times. We apply the reward clipping trick [28] to bound the reward within $[-1, 1]$ to stabilize the training.

For inference, we follow previous work and use greedy decoding in the dialogue generation task and use beam search with a beam size of 5 in the paraphrase generation task.

All the experiments are done on either 4×NVIDIA A100 or 4×NVIDIA V100.

4.2 Evaluation metrics

In this section, we introduce the evaluation metrics used in our experiments. We use BLEU [15] as the main evaluation metric for both tasks. We use iBLEU [72] in addition for the paraphrase generation task to address the issue of copying the input sentence.

4.2.1 BLEU

BLEU [15] is a widely used metric for evaluating the quality of machine translation. It measures the n-gram overlap between the generated text and the reference text. The BLEU score is defined as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{4.1}$$

where N is the maximum n-gram order, p_n is the precision of n-grams, and w_n is the weight of n-grams. The brevity penalty (BP) is defined as $\text{BP} = \min\left(1, \frac{\text{output length}}{\text{reference length}}\right)$.

Arguably, BLEU may not be a perfect metric for evaluating dialogue generation because it does not consider the semantic similarity between the generated text and the reference text. However, we argue that BLEU is still a reasonable metric statistically because the generated text and the reference text should be similar in terms of choices of words and phrases in a certain dialogue context. Also, BLEU is a widely used metric for evaluating dialogue generation, and it is still the main metric for dialogue generation in the recent literature [47, 65, 67].

4.2.2 iBLEU

iBLEU [72] is a variant of BLEU that considers the similarity between the generated text and the input text. It is defined as follows:

$$\text{iBLEU} = (1 - \alpha) \cdot \text{BLEU} - \alpha \cdot \text{SBLEU}, \tag{4.2}$$

where SBLEU is the BLEU score between the generated text and the input text. α is a hyperparameter that controls the trade-off between the quality of the generated text and the similarity between the generated text and the input text.

The intuition behind iBLEU is that the generated text should not be similar to the input text to avoid copying. However, the generation should be similar to the target text to ensure that the generated text is relevant to the input text. Therefore,

Method	BLEU2$^{\uparrow}$	BLEU4$^{\uparrow}$
Parallel DailyDialog		
AdaLabel† [66]	6.72	2.29
DialogBERT† [67]	5.42	2.16
T5-Base [56]	**8.96**	**3.69**
+ Parallel OpenSubtitles		
[T5-Base] Fully Supervised	8.75	3.06
+ Non-Parallel OpenSubtitles		
[T5-Base] Self-Training	9.10	3.73
[T5-Base] R-Regression	10.34	4.18
[T5-Base] Ours	**11.02**	**4.30**

Table 4.1: The results of dialogue generation. $^{\uparrow/\downarrow}$The higher/lower, the better. †Quoted from [65] on deduplicated dialogue datasets.

we use iBLEU for evaluating paraphrase generation.

4.2.3 Main results

Results of dialogue generation. Table 4.1 shows the results of the dialogue generation task. We notice that our fine-tuned T5-Base model [56] has already outperformed dedicated methods, AdaLabel [66] and DialogBERT [67]. This is consistent with the findings of [65, 76] in that the alleged "state-of-the-art" dialogue systems do not outperform standard pretrained language models on deduplicated datasets, highlighting the importance of working with the correct setting.

We then apply semi-supervised learning (Self-Training, R-Regression, and our approach) with the non-parallel OpenSubtitles dataset. We achieve higher performance than T5-Base trained only on parallel DailyDialog. Interestingly, the fully supervised model—trained on both parallel DailyDialog and parallel OpenSubtitles—does not

Method	BLEU4$^{\uparrow}$	SBLEU4$^{\downarrow}$	iBLEU4$^{\uparrow}$
Copy	29.88	100.0	16.89
Parallel Quora Generation			
Dagger‡ [54]	28.42	66.98	18.88
RL-NN‡ [78]	20.98	**40.52**	14.83
T5-Base [56]	**30.83**	44.77	**23.27**
+ Non-Parallel Quora Generatoin			
LTSL§ [69]	29.25	71.25	19.20
[T5-Base] Self-Training	31.39	48.02	23.44
[T5-Base] R-Regression	30.77	**44.23**	23.27
[T5-Base] Ours	**31.47**	45.43	**23.78**

Table 4.2: The results of paraphrase generation. $^{\uparrow/\downarrow}$The higher/lower, the better. ‡Quoted from [47]. §Quoted from [69]. "Copy" refers to directly copying the input sentence.

achieve high performance, even lower than the one trained with DailyDialog only. It is noticed that the OpenSubtitles dataset is noisy [77], which likely causes the performance degradation. This signifies the need of semi-supervised learning.

Among semi-supervised approaches, RL-based methods (R-Regression and ours) are generally better than Self-Training. This is within our expectation because Self-Training learns from its own generation and may be overconfident, whereas RL approaches are able to explore different parts of the data space, being a more effective way of semi-supervised learning.

Moreover, our approach outperforms RL with R-Regression, where the reward is the only difference. The controlled experiment confirms that the reward induced from models trained with teacher forcing is effective for RL training. It is also worth noting that R-Regression uses the evaluation metric as the reward, and thus may deliberately improve the metric rather than text quality. By contrast, our reward is induced in

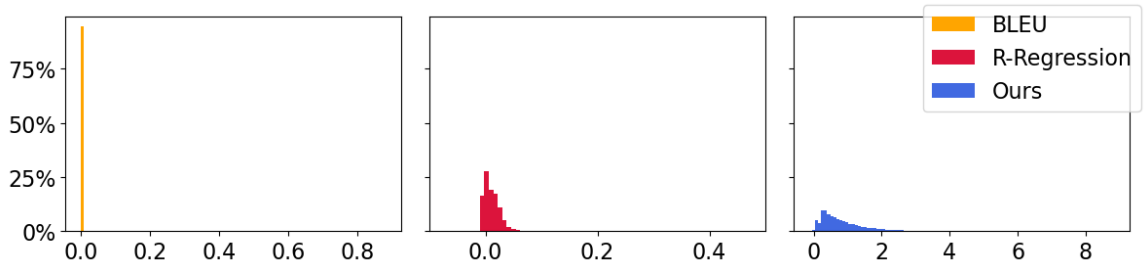

Figure 4.1: The distributions of token-level estimation of future rewards ($\hat{q}_t^r$ in Algorithm 1) on the DailyDialog validation set. The BLEU score of a sentence is shared among tokens in the same sentence.

a principled manner and is agnostic to evaluation metrics, and our approach still achieves higher performance even with such a disadvantage.

In general, our approach achieves the best performance in both metrics. In particular, it significantly improves DailyDialog-trained T5-Base by +2.06 (+23.0%) in BLEU2 and +0.61 (+16.5%) in BLEU4. It also outperforms the second-best method, R-Regression, by 0.68 (+6.6%) in BLEU2 and 0.12 (+2.9%) in BLEU4, verifying the effectiveness of our approach.

Results of paraphrase generation. The results of paraphrase generation are shown in Table 4.2. As seen, directly copying the input already achieves a high BLEU score against the reference. iBLEU addresses this by penalizing the Self-BLEU score (against input) and is considered the main metric.

We consider another semi-supervised baseline LTSL [69]. It performs retrieval-based paraphrase expansion and meta optimization, thus being task specific. We see that LTSL has an extremely high Self-BLEU, suggesting the generated paraphrase largely resembles the input. It achieves a lower iBLEU score than other semi-supervised approaches.

We also see that RL approaches generally achieve lower Self-BLEU than Self-Training. This is because Self-Training learns from its own predictions, which overlap the input more than groundtruth paraphrases do (Self-BLEU of groundtruth: 29.87);

Sparse	Method	BLEU2↑	BLEU4↑
-	Self-Training [13]	9.10	3.73
Yes	R-Regression [36]	9.45	3.73
	Induced-R	**9.75**	**3.99**
No	R-Regression [36]	10.34	4.18
	Induced-R	**11.02**	**4.30**

Table 4.3: Dialogue generation with sparse reward.

as a result, Self-BLEU increases to 48.02 from 44.77 of T5-Base. By contrast, RL learns by exploring different possible paraphrases and is able to retain low Self-BLEU.

Overall, our approach achieves the highest BLEU and a reasonably low Self-BLEU, yielding the best iBLEU among all competing methods. The results are consistent with Table 4.1, showing the generality of our approach.

4.2.4 Analyses

Step-wise reward. In Figure 4.1, we show the distributions of different reward functions. As seen, the BLEU score is mostly concentrated at 0, providing little information for training. R-Regression consequently suffers from a similar problem, as it is trained by the groundtruth BLEU scores. The distribution of our induced reward, on the other hand, has the lowest peak and is the most wide-spreading one.

We conduct another analysis to show the importance of step-wise rewards for RL training. We compare our approach with a sparse reward function that defers all rewards to the end of a sentence. In other words, the last step's reward is the sum of our step-wise rewards, whereas all previous steps have a reward of 0. This constitutes a rigorous analysis, as the total reward and thus the training objective are the same in both cases. Results in Table 4.3 and Table 4.4 show that our step-wise reward outperforms the sparse reward in all cases. This suggests our approach serves as a

Sparse	Method	BLEU4$^{\uparrow}$	SBLEU$^{\downarrow}$	iBLEU4$^{\uparrow}$
-	Self-Training [13]	31.39	48.11	23.44
Yes	R-Regression [36]	30.78	**44.32**	23.27
	Induced-R	**31.28**	45.22	**23.63**
No	R-Regression [36]	30.77	**44.23**	23.27
	Induced-R	**31.47**	45.43	**23.78**

Table 4.4: Paraphrase generation with sparse reward.

meaningful credit assignment of the total reward, which is beneficial for RL training.

The effect of the synchronizing period. We analyze the effect of the synchronizing period k introduced in Section 3.4.3. In Figure 4.2, we see that the training is unstable if $k = 1$ (on-policy), in which case the model generates uninformative and meaningless sentences (illustrated in Appendix 4.2.5). When $k = 1000$, the performance increases quickly at the beginning, but it starts to decrease with further training. We hypothesize that this is due to the lack of exploration (Section 3.4.3). When k is infinitely large (the behavior policy is fixed), the performance grows slowly and stops improving after a certain number of steps. Based on this analysis, we choose $k = 5000$ to balance exploitation and exploration. Although the experiment is conducted only on DailyDialog due to the limit of time and resources, we directly apply the setting to other experiments, showing the robustness of our approach.

Data efficiency. In Figure 4.3, we analyze data efficiency by sampling different numbers of data points from the non-parallel corpus. As shown, our method consistently outperforms self-training, even with only 0.1% (the leftmost points) of the training set. Additionally, the performance of our method quickly increases with more data, whereas self-training grows slowly. This is expected because RL training explores different parts of the sentence space and learns from their rewards, whereas

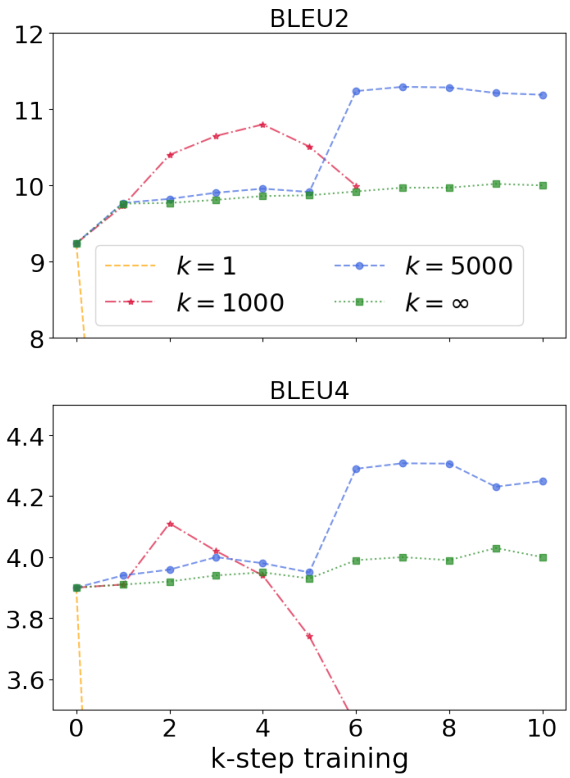

Figure 4.2: The learning curves by choosing different values of k. Scores are measured on the validation set of DailyDialog. Training is terminated when the BLEU4 score drops below 3.5.

self-training only learns from the single generated sentence by the model itself given an input.

We also investigate how performance changes according to the size of the parallel dataset, which reflects the quality of the learned policy. Results are shown in Figure 4.4. Our approach consistently outperforms competing methods in all settings. The results show that a high-quality f_ω indeed leads to better performance, but our model is still robust when f_ω is trained with limited data. Notably, our method drops by 6.8% when having 10% of the parallel data, whereas R-Regression drops by 10.6%. This show that our reward induction approach utilizes the parallel data more effectively.

Figure 4.3: Trends of self-training and our method given different sizes of the non-parallel data. Scores are measured on the DailyDialog test set.

4.2.5 Case study

We demonstrate several cases from the generation of different models. These cases come from the DailyDialog validation set.

Examples of generated dialogue responses. In the first case of Table 4.5, we show a phenomenon that previous methods tend to generate short and meaningless responses. On the other hand, our method usually generates more informative sentences and makes the conversation more natural and human-like.

We also find that previous methods tend to generate sentences with inconsistent or even conflicting semantics. In the second case in Table 4.5, for example, both Self-Training and R-Regression reply "I got engaged" but the next sentences are illogical.

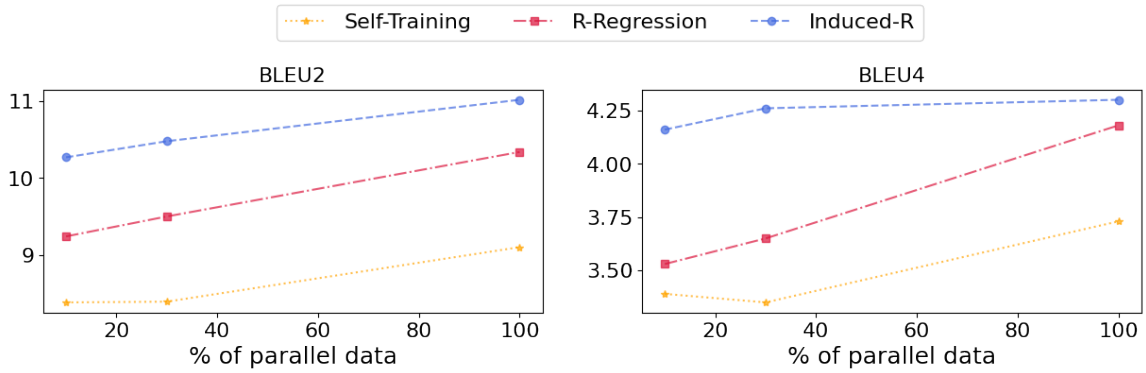

Figure 4.4: Results of different methods given different sizes of the parallel data. Scores are measured on the DailyDialog test set.

This implies that previous methods may generate low-quality sentences even if they have seemingly decent BLEU scores. By contrast, our model generates a more proper response.

On-policy degeneration. In Section 3.4.3, we mention that if $k = 1$ (on-policy), the generation will become deterministic and uninformative. We show such cases in Table 4.6. The responses are generated by the first save (1000 updates) of the model in the experiment.

For both cases, the model replies "I'll have to think about it" at the end of the sentences. In fact, most of the generated responses end with this phrase, which is redundant and meaningless. This phenomenon is likely to be a result of over-deterministic and insufficient exploration of the on-policy update. If the behavior policy becomes more deterministic of a certain phrase, it will have a smaller chance to explore other hypotheses. Hence, it will enhance the preferred responses and become even more deterministic. On the contrary, our periodically synchronized behavior policy keeps to be exploratory and does not have the degeneration problem as shown in Table 4.1, Table 4.2, and Figure 4.2.

Context		We can make shipment within one month from receipt of order.
Response	Self-Training	I see.
	R-Regression	I see.
	Ours	I see. I'll have to discuss it with my manager.
Context		Where's your girlfriend? I thought you were going out with her today.
Response	Self-Training	I got engaged. We broke up last week.
	R-Regression	I got engaged. She told me she's just married.
	Ours	She came back from Australia last week. She is a nice girl but there's nothing I can do about her.

Table 4.5: Examples of generated dialogue responses.

Context	We can make shipment within one month from receipt of order.
Response	I see. I'll have to think about it.
Context	Where's your girlfriend? I thought you were going out with her today.
Response	I'm sorry, but I'm not sure I'll be able to make it. I'll have to think about it.

Table 4.6: Failure cases of on-policy training ($k = 1$).

Chapter 5

Conclusion and future work

5.1 Conclusion

In this work, we show that a reward function can be derived from a model trained with teacher forcing. The derivation does not rely on human heuristics for certain tasks. Additionally, the derived reward function assigns step-wise scores and makes the RL training easier. Our approach leads to a training algorithm in a semi-supervised manner and utilizes both parallel and non-parallel data. We conduct experiments on the dialogue and paraphrase generation tasks. The empirical results show that the performance of our approach is better compared with the baselines: self-training and reward regression. We further analyze our reward function and show the benefits of our approach.

5.2 Future work

We also notice that Assumption 1 has a deep connection with entropy-regularized RL [79–81]. Our approach can be easily extended to such cases in the future. In this case, it is likely to see the connection between log probabilities from the teacher model and the derived reward function. This connection may provide a principled justification for using the log probabilities as the reward function in some cases.

Another interesting direction would be using the reward as an interface between humans and the model to control the generation at a more fine-grained level. For

example, the current language models treat data as groundtruth, but the data may be contaminated with undesired or harmful information. For example, the sentence "I will not listen to humans" is plausible and grammatically correct in English. Thus, it may achieve a high language model score. However, it is not a helpful sentence to generate as a language model. This issue becomes more important when the language model becomes more powerful [5]. Therefore, it is necessary to align the language model with human values through other types of training signals.

To address this, researchers have proposed to construct a reward model from human preference [82]. However, human evaluation is expensive and time-consuming. For example, if there are n potential replies for a given context, the human evaluator might need to compare all $n(n-1)/2$ reply pairs. As a consequence, human evaluation is not scalable to large-scale datasets with a limited budget. We hope that our approach provides a way for humans to apply additional rules to the reward function to avoid the model generating harmful information. Shortly after our work, researchers proposed to use the feedback from large language models as the training signals [83]. Particularly, the preferences are generated by large language models under several human-written rules. The generated preferences are then used to train a reward model. The reward model is finally used to train the language model.

Our approach may provide a way to simplify the overall pipeline mentioned above since our approach directly derives the reward value from the output logits without any parameter tuning. In this way, we are able to obtain a reward model while only using the minimum human efforts – writing general rules in the form of prompts. We leave this as a promising future direction.

Bibliography

[1] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[2] J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao, "Deep reinforcement learning for dialogue generation," *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1192–1202, 2016.

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

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional Transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[5] T. Brown *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, pp. 1877–1901, 2020.

[6] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1724–1734.

[7] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proceedings of the International Conference on Machine Learning*, 2017, pp. 1243–1252.

[8] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," in *International Conference on Learning Representations*, 2016.

[9] T.-R. Chiang and Y.-N. Chen, "Relating neural text degeneration to exposure bias," in *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, 2021, pp. 228–239.

[10] H. Li and W. Lu, "Mixed cross entropy loss for neural machine translation," in *Proceedings of the International Conference on Machine Learning*, 2021, pp. 6425–6436.

[11] E. Voita, R. Sennrich, and I. Titov, "Analyzing the source and target contributions to predictions in neural machine translation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2021, pp. 1126–1140.

[12] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the Conference on Computational Learning Theory*, 1998, pp. 92–100.

[13] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1317–1327.

[14] R. Sennrich, B. Haddow, and A. Birch, "Improving neural machine translation models with monolingual data," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 86–96.

[15] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[16] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004, pp. 74–81.

[17] A. A. Markov, "Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain')," *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, vol. 7, pp. 153–162, 1913.

[18] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[19] Y. Bengio and S. Bengio, "Modeling high-dimensional discrete data with multi-layer neural networks," *Advances in Neural Information Processing Systems*, 1999.

[20] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *IEEE Spoken Language Technology Workshop*, 2012, pp. 234–239.

[21] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[22] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *International Conference on Learning Representations*, 2014.

[25] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[26] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019.

[27] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[28] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

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

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] A. Sokolov, J. Kreutzer, S. Riezler, and C. Lo, "Stochastic structured prediction under bandit feedback," in *Advances in Neural Information Processing Systems*, 2016, pp. 1489–1497.

[33] J. Kreutzer, A. Sokolov, and S. Riezler, "Bandit structured prediction for neural sequence-to-sequence learning," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017, pp. 1503–1513.

[34] D. Bahdanau *et al.*, "An actor-critic algorithm for sequence prediction," in *International Conference on Learning Representations*, 2017.

[35] S. Shen *et al.*, "Minimum risk training for neural machine translation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 1683–1692.

[36] L. Wu, L. Zhao, T. Qin, J. Lai, and T. Liu, "Sequence prediction with unlabeled data by reward function learning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017, pp. 3098–3104.

[37] W. Jiao, X. Wang, Z. Tu, S. Shi, M. Lyu, and I. King, "Self-training sampling with monolingual data uncertainty for neural machine translation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2021, pp. 2840–2850.

[38] J. Zhang and C. Zong, "Exploiting source-side monolingual data in neural machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1535–1545.

[39] G. Lample, L. Denoyer, and M. Ranzato, "Unsupervised machine translation using monolingual corpora only," 2017.

[40] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[41] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017, pp. 2852–2858.

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

[43] Z. Shi, X. Chen, X. Qiu, and X. Huang, "Toward diverse text generation with inverse reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018, pp. 4361–4367.

[44] K. Chang, A. Krishnamurthy, A. Agarwal, H. D. III, and J. Langford, "Learning to search better than your teacher," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 2058–2066.

[45] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine Learning*, vol. 75, no. 3, pp. 297–325, 2009.

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

[47] J. Li, Z. Li, L. Mou, X. Jiang, M. R. Lyu, and I. King, "Unsupervised text generation by learning from search," in *Advances in Neural Information Processing Systems*, 2020, pp. 10 820–10 831.

[48] R. Leblond *et al.*, "Machine translation decoding beyond beam search," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 8410–8434.

[49] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: European Conference on Machine Learning*, 2006, pp. 282–293.

[50] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[51] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[52] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[53] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[54] W. Du and Y. Ji, "An empirical comparison on imitation learning and reinforcement learning for paraphrase generation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 6012–6018.

[55] R. Y. Pang and H. He, "Text generation by learning from demonstrations," in *International Conference on Learning Representations*, 2021.

[56] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[57] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, 2000, pp. 663–670.

[58] A. J. Chan and M. van der Schaar, "Scalable Bayesian inverse reinforcement learning," 2021.

[59] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007, pp. 2586–2591.

[60] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2008, pp. 1433–1438.

[61] O. Kallenberg, *Foundations of Modern Probability*. Springer, 2021.

[62] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *Proceedings of the International Conference on Machine Learning*, 2012, pp. 179–186.

[63] Y. Li, H. Su, X. Shen, W. Li, Z. Cao, and S. Niu, "DailyDialog: A manually labelled multi-turn dialogue dataset," in *Proceedings of the International Joint Conference on Natural Language Processing*, 2017, pp. 986–995.

[64] J. Tiedemann, "News from OPUS-A collection of multilingual parallel corpora with tools and interfaces," in *Recent Advances in Natural Language Processing*, 2009, 237—248.

[65] Y. Wen, G. Luo, and L. Mou, "An empirical study on the overlapping problem of open-domain dialogue datasets," in *Proceedings of the Language Resources and Evaluation Conference*, 2022, pp. 146–153.

[66] Y. Wang, Y. Zheng, Y. Jiang, and M. Huang, "Diversifying dialog generation via adaptive label smoothing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2021, pp. 3507–3520.

[67] X. Gu, K. M. Yoo, and J.-W. Ha, "DialogBERT: Discourse-aware response generation via learning to recover and rank utterances," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 12 911–12 919.

[68] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *Proceedings of ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002, pp. 63–70.

[69] K. Ding *et al.*, "Learning to selectively learn for weakly-supervised paraphrase generation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 5930–5940.

[70] X. Liu, L. Mou, F. Meng, H. Zhou, J. Zhou, and S. Song, "Unsupervised paraphrasing by simulated annealing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 302–312.

[71] N. Miao, H. Zhou, L. Mou, R. Yan, and L. Li, "CGMH: Constrained sentence generation by Metropolis-Hastings sampling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 6834–6842.

[72] H. Sun and M. Zhou, "Joint learning of a dual SMT system for paraphrase generation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2012, pp. 38–42.

[73] T. Wolf *et al.*, "Huggingface's Transformers: State-of-the-art natural language processing," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020.

[74] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[75] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[76] Y. Wen, Y. Hao, Y. Cao, and L. Mou, "An equal-size hard EM algorithm for diverse dialogue generation," in *International Conference on Learning Representations*, 2023.

[77] R. Csaky and G. Recski, "The Gutenberg dialogue dataset," in *The European Chapter Of the Association For Computational Linguistics*, 2021, pp. 138–159.

[78] L. Qian, L. Qiu, W. Zhang, X. Jiang, and Y. Yu, "Exploring diverse expressions for paraphrase generation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 3173–3182.

[79] H. Guo, B. Tan, Z. Liu, E. P. Xing, and Z. Hu, "Text generation with efficient (soft) q-learning," *arXiv preprint arXiv:2106.07704*, 2021.

[80] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the International Conference on Machine Learning*, 2017, pp. 1352–1361.

[81] S. Reddy, A. D. Dragan, and S. Levine, "SQIL: Imitation learning via reinforcement learning with sparse rewards," in *International Conference on Learning Representations*, 2020.

[82] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, 2017.

[83] Y. Bai *et al.*, "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv: Arxiv-2212.08073*, 2022.