Deep Reinforcement Learning for Sequence-to-Sequence Models

Yaser Keneshloo, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy, *Senior Member, IEEE*

***Abstract*— In recent times, sequence-to-sequence (seq2seq) models have gained a lot of popularity and provide state-of-the-art performance in a wide variety of tasks, such as machine translation, headline generation, text summarization, speech-to-text conversion, and image caption generation. The underlying framework for all these models is usually a deep neural network comprising an encoder and a decoder. Although simple encoder–decoder models produce competitive results, many researchers have proposed additional improvements over these seq2seq models, e.g., using an attention-based model over the input, pointer-generation models, and self-attention models. However, such seq2seq models suffer from two common problems: 1) exposure bias and 2) inconsistency between train/test measurement. Recently, a completely novel point of view has emerged in addressing these two problems in seq2seq models, leveraging methods from reinforcement learning (RL). In this survey, we consider seq2seq problems from the RL point of view and provide a formulation combining the power of RL methods in decision-making with seq2seq models that enable remembering long-term memories. We present some of the most recent frameworks that combine the concepts from RL and deep neural networks. Our work aims to provide insights into some of the problems that inherently arise with current approaches and how we can address them with better RL models. We also provide the source code for implementing most of the RL models discussed in this paper to support the complex task of abstractive text summarization and provide some targeted experiments for these RL models, both in terms of performance and training time.**

***Index Terms*— Actor–critic (AC) methods, deep learning, policy gradients (PGs), *Q*-learning, reinforcement learning (RL), sequence-to-sequence (seq2seq) learning.**

I. Introduction

SEQUENCE-TO-SEQUENCE (seq2seq) models constitute a common framework for solving sequential problems [1]. In seq2seq models, the input is a sequence of certain data units, and the output is also a sequence of data units. Traditionally, these models are trained using a ground-truth sequence via a mechanism known as *teacher forcing* [2], where the teacher is the ground-truth sequence. However, due to some of the drawbacks of this training approach, there has been a significant line of research connecting inference of these models with reinforcement learning (RL) techniques. In this paper, we aim to summarize such research in seq2seq training utilizing the RL methods to enhance the performance of these models and discuss various challenges that arise when applying the RL methods to train a seq2seq model. We intend for this paper to provide a broad overview of the strength and complexity of combining seq2seq training with RL training and to guide researchers in choosing the right RL algorithm for solving their problem. In this section, we will briefly introduce the working of a simple seq2seq model and outline some of the problems that are inherent to seq2seq models. We will then provide an introduction to RL models and explain how these models could solve the problems of seq2seq models.

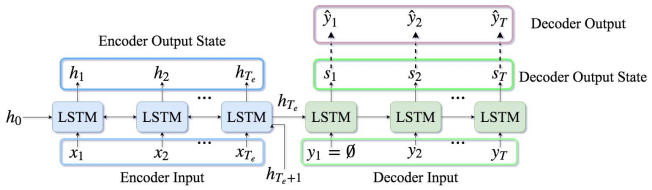

Fig. 1. Simple seq2seq model. The blue boxes correspond to the encoder part that has T_e units. The green boxes correspond to the decoder part that has T units.

A. seq2seq Framework

The seq2seq models are common in various applications ranging from machine translation [3]–[8], news headline generation [9], [10], text summarization [11]–[14], speech-to-text applications [15]–[18], and image captioning [19]–[21].

In recent years, the general framework for solving these problems uses deep neural networks that comprise two main components: an encoder that reads the sequence of input data and a decoder that uses the output generated by the encoder to produce the sequence of final outputs. Fig. 1 gives a schematic of this simple yet effective framework. The encoder and decoder are usually implemented by recurrent neural networks (RNNs), such as long short-term memory (LSTM) [22]. The encoder takes a sequence of length T_e inputs,[1] $X = \{x_1, x_2, \cdots, x_{T_e}\}$, where $x_t \in \mathcal{A} = \{1, \ldots, |\mathcal{A}|\}$ is a single input coming from a range of possible inputs ($\mathcal{A}$), and generates the output state h_t. In addition, each encoder receives the previous encoder's hidden state, h_{t-1}, and if the encoder is a bidirectional RNN, it will also receive the state from the next encoder's hidden state, h_{t+1}, to generate its current hidden state h_t. The decoder, on the other hand, takes the last state from the encoder, i.e., h_{T_e} and starts generating an output of size $T \leq T_e$, $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T\}$, based

Manuscript received July 28, 2018; revised December 27, 2018 and April 19, 2019; accepted July 2, 2019. This work was supported in part by the U.S. National Science Foundation under Grant IIS-1619028, Grant IIS-1707498, Grant IIS-1838730, Grant DGE-1545362, and Grant IIS-1633363. *(Corresponding author: Yaser Keneshloo.)*

The authors are with the Discovery Analytics Center, Department of Computer Science, Virginia Tech, Arlington, VA 22203 USA (e-mail: yaserkl@vt.edu; tshi@vt.edu; naren@cs.vt.edu; reddy@cs.vt.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2929141

[1] In this article, we use input/output and action interchangeably since choosing the next input is akin to choosing the next action and generating the next output is akin to generating the next action.

on the current state of the decoder s_t and the ground-truth output y_t. The decoder could also take as input an additional context vector c_t that encodes the context to be used while generating the output [9]. The RNN learns a recursive function to compute s_t and outputs the distribution over the next output

$$\begin{aligned} h_{t+1} &= \Phi_\theta(x_{t+1}, h_t) \\ s_{t+1} &= \Phi_\theta(y_t, s_t/h_{T_e}, c_t) \\ \hat{y}_{t+1} &\sim \pi_\theta(y|\hat{y}_t, s_{t+1}) \end{aligned} \tag{1}$$

where θ denotes the parameters of the model, and the function for π_θ and Φ_θ depends on the type of RNN. A simple Elman RNN [23] would use a sigmoid function for Φ and a softmax function for π [1]

$$\begin{aligned} s_{t+1} &= \sigma(W_1 y_t + W_2 s_t + W_3 c_t) \\ o_{t+1} &= \text{softmax}(W_4 s_{t+1} + W_5 c_t) \end{aligned} \tag{2}$$

where o_t is the output distribution of size $|\mathcal{A}|$ and the output $\hat{y}_t$ is selected from this distribution. W_1–W_5 are the matrices of learnable parameters of sizes $W_{1,2,3} \in R^{d\times d}$ and $W_{4,5} \in R^{d\times|\mathcal{A}|}$, where d is the size of the input representation (e.g., size of the word embedding in text summarization). The input to the first decoder is a special input indicating the beginning of a sequence, denoted by $y_0 = \emptyset$, and the first forward hidden state h_0 and the last backward hidden state h_{T_e+1} for the encoder are set to a zero vector. Moreover, the first hidden state for decoder s_0 is set to the output that is received from the last encoding state, i.e., h_{T_e}.

The most widely used method to train the decoder for sequence generation is called the teacher forcing algorithm [2], which minimizes the maximum-likelihood loss at each decoding step. Let us define $y = \{y_1, y_2, \ldots, y_T\}$ as the ground-truth output sequence for a given input sequence X. The maximum-likelihood training objective is the minimization of the following cross-entropy (CE) loss:

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log \pi_\theta(y_t|y_{t-1}, s_t, c_{t-1}, X). \tag{3}$$

Once the model is trained with the above-mentioned objective, the model generates an entire sequence as follows: Let $\hat{y}_t$ denote the action (output) taken by the model at time t. Then, the next action is generated by

$$\hat{y}_{t+1} = \arg\max_y \pi_\theta(y|\hat{y}_t, s_{t+1}). \tag{4}$$

This process could be improved by using beam search to find a reasonable good output sequence [7]. Now, given the ground-truth output Y and the model generated output $\hat{Y}$, the performance of the model is evaluated with a specific measure. In seq2seq problems, discrete measures, such as ROUGE [24], BLEU [25], METEOR [26], and CIDEr [27], are used to evaluate the model. For instance, ROUGE$_l$, an evaluation measure for textual seq2seq tasks, uses the largest common substring (LCS) between Y and $\hat{Y}$ to evaluate the goodness of the generated output. Algorithm 1 shows these steps.

Algorithm 1 Training a Simple seq2seq Model

Input: Input sequences (X) and ground-truth output sequences (Y).
Output: Trained seq2seq model.
Training Steps:
for batch of input and output sequences X and Y **do**
  Run encoding on X and get the last encoder state h_{T_e}.
  Run decoding by feeding h_{T_e} to the first decoder and obtain the sampled output sequence $\hat{Y}$.
  Calculate the loss according to Eq. (3) and update the parameters of the model.
end for
Testing Steps:
for batch of input and output sequences X and Y **do**
  Use the trained model and Eq. (4) to sample the output $\hat{Y}$
  Evaluate the model using a performance measure, e.g., ROUGE
end for

B. Problems With seq2seq Models

One of the main issues with the current seq2seq models is that minimizing $\mathcal{L}_{\text{CE}}$ does not always produce the best results for the above-mentioned discrete evaluation measures. Therefore, using CE loss for training a seq2seq model creates a mismatch in generating the next action during training and testing. As shown in Fig. 1 and also according to (3), during training, the decoder uses the two inputs, the previous output state s_{t-1} and the ground-truth input y_t, to calculate its current output state s_t and uses it to generate the next action, i.e., $\hat{y}_t$. However, at the test time, as given in (4), the decoder completely relies on the previously generated action from the model distribution to predict the next action since the ground-truth data are not available anymore. Therefore, in summary, the input to the decoder is from the ground truth during training, but the input comes from the model distribution during model testing. This *exposure bias* [28] results in error accumulation during the output generation at test time since the model has never been exclusively exposed to its own predictions during training. To avoid the *exposure bias* problem, we need to remove the ground-truth dependency during training and use only the model distribution to minimize (3). One way to handle this situation is through the scheduled sampling method [2] or the Gibbs sampling [29]. In scheduled sampling, the model is first pre-trained using the CE loss and will subsequently and slowly replace the ground truth with a sampled action from the model. Therefore, a decision is randomly taken to whether to use the ground-truth action with probability ϵ or an action coming from the model itself with probability $(1-\epsilon)$. When $\epsilon = 1$, the model is trained using (3), and when $\epsilon = 0$, the model is trained based on the following loss:

$$\mathcal{L}_{\text{Inference}} = -\sum_{t=1}^{T} \log \pi_\theta(\hat{y}_t|\hat{y}_1, \cdots, \hat{y}_{t-1}, s_t, c_{t-1}, X). \tag{5}$$

Note the difference between $L_{\text{Inference}}$ and the CE loss given in (3); in CE, the ground-truth output y_t is used to calculate the loss, while in (5), the output of the model $\hat{y}_t$ is used to calculate the loss.

Although scheduled sampling is a simple way to avoid the exposure bias, due to its random selection between choosing

the ground-truth output or the model output, it does not provide a clear solution for the backpropagation of error, and therefore, it is statistically inconsistent [30]. Recently, Goyal *et al.* [31] proposed a solution for this problem by creating a continuous relaxation over the argmax operation to create a differentiable approximation of the greedy search during the decoding steps.

As yet another line of research on avoiding the exposure bias problem, adversarial generative models are also proposed for various seq2seq models [32]–[35]. In general, adversarial models are comprised of a discriminator and a generator [36]. The generator tries to generate the data similar to the ground-truth data, while the discriminator's job is to discern whether the generated data are close to the real data or it is a fake. Finally, the generator takes the feedback from the discriminator and optimizes its actions toward generating higher quality data. Since generator will only rely on its own output in generating the data, similar to the scheduled sampling, it is avoiding on reliance to the ground-truth data and, hence, avoids the exposure bias problem. However, adversarial generative models, in general, suffer from the reward sparsity [33], [35] and mode collapse [37] problems. Although there are ways to avoid these two problems [38], studying these solutions is outside the scope of this paper.

The second problem with seq2seq models is that while the model training is done using $\mathcal{L}_{\text{CE}}$, the model is typically evaluated during the test time using discrete and non-differentiable measures, such as BLEU and ROUGE. This will create a mismatch between the training objective and the test objective and, therefore, could yield inconsistent results. Thus, a solution that could use these measures during the training of the model will inherently solve this mismatch problem. Recently, it has been shown that both the exposure bias and non-differentiability of evaluation measures can be addressed by incorporating techniques from RL [13], [39]–[41].

C. Reinforcement Learning

In RL, a sequential Markov decision process (MDP) is considered, in which an agent interacts with an environment ε over discrete time steps t [42]. Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, s_0, \gamma, T)$ represent this discrete finite-horizon discounted MDP, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is the transition probability distribution, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $s_0 : \mathcal{S} \to \mathbb{R}_+$ is the initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T is the horizon.

The goal of the agent is to excel at a specific task, e.g., moving an object [43], [44], playing an Atari game [45], or generating news summary [13], [46]. The idea is that given the environment state at time t as s_t, the agent picks an action $\hat{y}_t \in \mathcal{A}$, according to a (typically stochastic) policy $\pi(\hat{y}_t|s_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ and observes a reward r_t for that action. The cumulative discounted sum of rewards is the objective function optimized by policy π. For instance, we can consider our seq2seq conditioned RNN as a stochastic policy that generates actions (selecting the next output) and receives the task reward based on the discrete measures, such as ROUGE, as the return. The agent's goal is to maximize the expected discounted reward, $R_t = \mathbb{E}_\pi[\sum_{\tau=0}^{T} \gamma^\tau r_\tau]$, where the discounting factor γ controls the tradeoffs between the importance of immediate and future rewards. Under the policy π, we can define the values of the state–action pair $Q(s_t, y_t)$ and the state $V(s_t)$ as follows:

$$\begin{aligned} Q_\pi(s_t, y_t) &= \mathbb{E}[r_t | s = s_t, y = y_t] \\ V_\pi(s_t) &= \mathbb{E}_{y\sim\pi(s)}[Q_\pi(s_t, y = y_t)]. \end{aligned} \tag{6}$$

Note that the value function V_π is defined over only the states, whereas Q_π is defined over (state–action) pairs. The Q_π formulation is advantageous in model-free contexts since it can be applied to the *current* states without having access to a model of the environment. In contrast, the V_π formulation must, by necessity, be applied to the *future* states and, thus, requires a model of the environment (i.e., which states and actions lead to which other future states). The preceding state–action function (Q-function for short) can be computed recursively with dynamic programming

$$Q_\pi(s_t, y_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \underbrace{\mathbb{E}_{y_{t+1}\sim\pi(s_{t+1})}[Q_\pi(s_{t+1}, y_{t+1})]}_{V_\pi(s_{t+1})}]. \tag{7}$$

Given the above-mentioned definitions, we can define a function called advantage, denoted by A_π relating the value function V and Q-function as follows:

$$\begin{aligned} A_\pi(s_t, y_t) &= Q_\pi(s_t, y_t) - V_\pi(s_t) \\ &= r_t + \gamma\, \mathbb{E}_{s_{t+1}\sim\pi(s_{t+1}|s_t)}[V_\pi(s_{t+1})] - V_\pi(s_t) \end{aligned} \tag{8}$$

where $\mathbb{E}_{y\sim\pi(s)}[A_\pi(s, y)] = 0$ and, for a deterministic policy, $y^* = \arg\max_y Q(s, y)$; it follows that $Q(s, y^*) = V(s)$, and hence, $A(s, y^*) = 0$. Intuitively, the value function V measures how good the model could be when it is in a specific state s. The Q-function, however, measures the value of choosing a specific action when we are in such state. Given these two functions, we can obtain the advantage function that captures the importance of each action by subtracting the value of the state V from the Q-function. In practice, seq2seq model is used as the policy, which generates actions. The definition of action, however, will be task-specific; e.g., for a text summarization task, and the action denotes choosing the next token for the summary, whereas for a question answering task, the action might be defined as the start and end index of the answer in the document. Also, the definition of the reward function could vary from one application to another. For instance, in text summarization, measures, such as ROUGE and BLEU, are commonly used, while in image captioning, CIDEr and METEOR are common. Finally, the state of the model is usually defined as the decoder output state at each time step. Therefore, the decoder output state at each time is used as the current state of the model and used to calculate our Q, V, and advantage functions. Table I summarizes the notations used in this paper.

D. Paper Organization

In general, we define the following problem statement that we are trying to solve by combining these two different models of learning.

Problem Statement: Given a series of input data and a series of ground-truth outputs, train a model that holds the following:

TABLE I
NOTATIONS USED IN THIS PAPER

Seq2seq Model Parameters	
X	The sequence of input of length T_e, $X = \{x_1, x_2, \cdots, x_{T_e}\}$.
Y	The sequence of ground-truth output of length T, $Y = \{y_1, y_2, \cdots, y_T\}$.
$\hat{Y}$	The sequence of output generated by model of length T, $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_T\}$.
T_e	Length of the input sequence and number of encoders.
T	Length of the output sequence and number of decoders.
d	Size of the input and output sequence representation.
$\mathcal{A}$	Input and output shared vocabulary.
h_t	Encoder hidden state at time t.
s_t	Decoder hidden state at time t.
π_θ	The seq2seq model with parameter θ.
Reinforcement Learning Parameters	
$r_t = r(s_t, y_t)$	The reward that the agent receives by taking action y_t when the state of the environment is s_t
$\hat{Y}$	Sets of actions that the agent is taking for a period of time T, $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_T\}$ This is similar to the output that the seq2seq model is generating.
π π_θ	The policy that the agent uses to take the action. Seq2seq models use RNNs with parameter θ for the policy.
γ	Discount factor to reduce the effect of rewards from future actions.
$Q(s_t, y_t)$ $Q_\pi(s_t, y_t)$	The Q-value (under policy π) that shows the estimated reward of taking action y_t when at state s_t.
$Q_\Psi(s_t, y_t)$	A function approximator with parameter Ψ that estimates the Q-value given the state-action pair at time t.
$V(s_t)$ $V_\pi(s_t)$	Value function which calculates the expectation of Q-value (under policy π) over all possible actions.
$V_\Psi(s_t)$	A function approximator with parameter Ψ that estimates the value function given the state at time t.
$A_\pi(s_t, y_t)$	Advantage function (under policy π) which defines how good a state-action pair is w.r.t. the expected reward that can be received at this state.
$A_\Psi(s_t, y_t)$	A function approximator with parameter Ψ that estimates the advantage function for the state-action pair at time t.

1) only relies on its own output, rather than the ground truth, to generate the results (avoiding exposure bias);
2) directly optimizes the model using the evaluation measure (avoiding a mismatch between training and test measures).

Although, recently, there had been a couple of survey articles on the topic of deep RL [47], [48], these works heavily focused on the RL methods and their applications in robotics and vision while giving less emphasis to how these models could be used in a variety of other tasks. In this paper, we will summarize some of the most recent frameworks that attempted to find a solution for the above-mentioned problem statement in a broad range of applications and explain how RL and seq2seq learning could benefit from each other in solving complex tasks. To this end, we will provide insights on some of the challenges and issues with the current models and how one can improve them with better RL models. The goal of this paper is to provide information about how we can broaden the power of seq2seq models with RL methods and understand the challenges that exist in applying these methods to deep learning contexts. In addition, currently, there does not exist a good open-source framework for implementing these ideas. Along with this paper, we provide a library that combines the state-of-the-art methods for the complex task of abstractive text summarization with recent techniques used in deep RL. The library provides a variety of different options and hyperparameters for training the seq2seq model using different RL models. Moreover, we provide experimental results on some of the most common techniques that are explained in this paper and we encourage researchers to experiment with other hyperparameters and explore how they can use this framework to gain better performance on different seq2seq tasks. The contributions of this paper are summarized as follows.

1) Provide a comprehensive summary of RL methods that are used in deep learning and specifically in the context of training seq2seq models.
2) Summarize the challenges, advantages, and disadvantages of using different RL methods for seq2seq training.
3) Provide guidelines on how one could improve a specific RL method to obtain a better and smoother training for seq2seq models.
4) Provide an open-source library for implementing a complex seq2seq model using different RL techniques[2] along with experiments that aim for identifying an accurate estimate on the amount of improvement that the RL algorithm provides for current seq2seq models.

This paper is organized as follows. Section II will cover some of the main applications of seq2seq models. Section III provides the details of some of the common RL techniques used in training seq2seq models. We provide a brief introduction to different seq2seq models in Section IV and later explain various RL models that can be used along with the seq2seq model training process. We provide a summary of recent real-world applications that combine RL training with seq2seq training, and in Section V, we present the framework that we implemented and discuss the details about how this framework can be applied to different seq2seq problems and provide experimental results for some of the well-known RL algorithm. Finally, in Section VI, we discuss the conclusions of our work.

II. SEQ2SEQ MODELS AND THEIR APPLICATIONS

seq2seq models have been an integral part of many real-world problems. From Google Machine Translation [4]

[2]www.github.com/yaserkl/RLSeq2Seq/

TABLE II
SUMMARY OF DIFFERENT APPLICATIONS OF SEQ2SEQ MODELS.

Application	Problem Description	Input	Output	References
Machine Translation	Translating a sentence from a source language to a target language	A sentence (sequence of words) in language X (e.g., English)	Another sentence (sequence of words) in language Y (e.g., French)	[1], [6], [7] [3], [50]
Text Summarization Headline Generation	Summarizing a document into a more concise and shorter text	A long document like a news article (sequence of words)	A short summary/headline (sequence of words)	[9], [10], [14], [51] [12], [52]–[57]
Question Generation	Generating interesting questions from a text document or an image	A piece of text (sequence of words) or image (sequence of layers)	A set of questions (sequence of words) related to the text or image	[58], [59]
Question Answering	Given a text document or an image and a question, find the answer to the question	A textual question (sequence of words) or an image (sequence of layers)	A single word answer from a document or the start and end index of the answer in the document	[60]–[62]
Dialogue Generation	Generate a dialogue between two agents e.g., between a robot and human	A dialogue from the first agent (sequence of words) or audibles (sequence of speech units)	A dialogue from the second agent (sequence of words) or audibles (sequence of speech units)	[63]–[65]
Semantic Parsing	Generating automatic SQL queries from a given human-written description	A human-written description of the query (sequence of words)	The SQL command equivalent to that description	[66], [67]
Image Captioning	Given an image, generate a caption that explains the content of the image	An image (sequence of layers)	The caption (sequence of words) describing that image	[68], [69] [19], [20], [70]
Video Captioning	Given a video clip, generate a caption that explains the content of the video	A video (sequence of images)	The caption (sequence of words) describing the video	[71]–[74]
Computer Vision	Finding interesting events in a video clip, e.g., predicting the next action of a specific object in the video	A video (sequence of images)	Differs from application to application. For instance, one might be interested in determining the next action of a specific object or entity in the video	[75]–[77]
Speech Recognition	Given a segment of audible input (e.g., speech), convert it to text and vice versa	A speech (sequence of speech units)	The text of the input speech (sequence of words)	[16], [17], [78]
Speech Synthesis	Given a segment of text, it generates its audible sounds	A text (sequence of words)	A speech representing (sequence of speech units) representing its audible sounds	[79], [80]

to Apple's Siri speech to text [49], seq2seq models provide a clear framework to process information that is in the form of sequences. In a seq2seq model, the input and output are in the form of sequences of single units, such as a sequence of words, images, or speech units. Table II provides a brief summary of various seq2seq models and their corresponding inputs and outputs. We also cite some of the important research papers for each application domain.

In recent years, different models and frameworks were proposed by researchers to achieve better and robust results on these tasks. For instance, attention-based models have been successfully applied to problems, such as machine translation [3], text summarization [9], [10], question answering [64], image captioning [19], speech recognition [16], and object detection [90]. In an attention-based model, at each decoding step, the previous decoder output is combined with the information from the encoder's output at a specific position to select the best decoder output.

Although attention-based models can significantly improve the performance of seq2seq models in various tasks, in applications with large output space, it is challenging for the model to reach a desirable outcome.

On the other hand, there are more advanced models in seq2seq training, such as the pointer-generator model [12], [91] and the transformers model, which uses self-attention layers [92], but discussing these models is outside the scope of this paper.

Aside from these well-defined seq2seq problems, there are other related problems that partially work on the sequence of inputs, but the output is not in the form of a sequence. Here are a few prominent applications that fall into this category.

1) *Sentiment Analysis [93]–[95]:* The input is a sequence of words, and the output is a single sentiment (positive, negative, or neutral).
2) *Natural Language Inference [96]–[98]:* Given two sentences, one as a premise and the other as a hypothesis, the goal is to classify the relationship between these two sentences into one of the entailment, neutrality, and contradiction classes.
3) *Sentiment Role Labeling [99]–[102]:* Given a sentence and a predicate, the goal is to answer questions, such as "who did what to whom and when and where."
4) *Relation Extraction [103]–[105]:* Given a sentence, the goal is to identify whether a specific relationship exists in that sentence or not. For instance, based on the sentence "Barack Obama is married to Michelle Obama," we can extract the "spouse" relationship.

5) *Pronoun Resolution [106]–[109]:* Given a sentence and a question about a pronoun in the sentence, the goal is to identify who that pronoun is referring to. For instance, in the sentence "Susan cleaned Alice's bedroom for the help **she** had given," the goal is to find who the word "she" is referring to.

Note that although in these applications, only the input data are represented in terms of sequences, we still consider them to be seq2seq problems.

A. Evaluation Measures

The seq2seq models are usually trained with the CE loss, i.e., (3). However, the performance of these models is evaluated using discrete measures. There are various discrete measures that are used for evaluating these models, and each application requires its own evaluation measure. We briefly provide a summary of these measures according to their application context.

1) *ROUGE*[3] *[24], BLEU*[4] *[25], and METEOR*[5] *[26]:* These are three of the most commonly used measures in applications, such as machine translation, headline generation, text summarization, question answering, dialog generation, and other applications, that require evaluation of text data. ROUGE measure finds the common unigram (ROUGE-1), bigram (ROUGE-2), and LCS (ROUGE-L) between the ground-truth text and the output generated by the model and calculates respective precision, recall, and F-score for each measure. BLEU works similar to ROUGE but through a modified precision calculation; it inclines to provide higher scores to outputs that are closer to human judgment. In a similar manner, METEOR uses the harmonic mean of unigram precision and recall, and it gives higher importance to recall than the precision. Although these methods are designed to work for all text-based applications, METEOR is more often used in machine translation tasks, while ROUGE and BLEU are mostly used in text summarization, question answering, and dialog generation.
2) *CIDEr*[6] *[27] and SPICE*[7] *[110]:* CIDEr is frequently used in image and video captioning tasks, in which having captions that have higher human judgment scores is more important. Using sentence similarity, the notions of grammaticality, saliency, importance, accuracy, precision, and recall are inherently captured by these metrics. SPICE is a recent evaluation metric proposed for image captioning that tries to solve some of the problems of CIDEr and METEOR by mapping the dependency parse trees of the caption to the semantic scene graph (contains objects, attributes of objects, and relations) extracted from the image. Finally, it uses the F-score that is calculated using the tuples of the generated and ground-truth scene graphs to provide the caption quality score.
3) *Word Error Rate (WER):* This measure, which is mostly used in speech recognition, finds the number of substitutions, deletions, insertions, and corrections required to change the generated output to the ground truth and combines them to calculate the WER.

B. Data Sets

In this section, we briefly describe some of the data sets that are commonly used in various seq2seq models. We provide a shortlist of some of the most common data sets that are used in various seq2seq applications as follows.

1) *Machine Translation:* The most common data set used for Machine Translation task is the WMT'14[8] data set that contains 850M words from English–French parallel corpora of UN (421M words), Europarl (61M words), news commentary (5.5M words), and two crawled corpora of 90M and 272.5M words. The data pre-processing for this data set is usually done following the code[9] provided by Axelrod *et al.* [111].
2) *Text Summarization:* One of the main data sets used in text summarization is the CNN-Daily Mail data set [112] that is a part of the DeepMind Q&A data set[10] and contains around 287k news articles along with two to four highlights (summary) for each news article.[11] Recently, another data set, called Newsroom, was released by Connected Experiences Lab[12] [113] that contains 1.3M news articles and various metadata information, such as the title and the summary of the news. The document summarization challenge[13] also provides some data sets for text summarization. More specifically, in this data set, DUC-2003 and DUC-2004 contain 500 news articles (each paired with four different human-generated reference summaries) from the New York Times and Associated Press Wire services, respectively. Due to the small size of this data set, researchers usually use this data set only for evaluation purposes.
3) *Headline Generation:* It is similar to the task of text summarization, and typically, all the data sets that are used in text summarization will be useful in headline generation too. There is a big data set, which is called Gigaword [114] and contains more than 8M news articles from multiple news agencies, such as The New York Times and Associated Press. However, this data set is not freely available, and researchers are required to buy the license to be able to use it though one can still find pre-trained models on different tasks using this data set.[14]
4) *Question Answering and Question Generation:* The CNN-Daily Mail data set was originally designed

[3]https://github.com/andersjo/pyrouge/
[4]https://www.nltk.org/_modules/nltk/translate/bleu_score.html
[5]http://www.cs.cmu.edu/ alavie/METEOR/
[6]https://github.com/vrama91/cider
[7]http://www.panderson.me/spice/

[8]http://www.statmt.org/wmt14/translation-task.html
[9]http://www-lium.univ-lemans.fr/ schwenk/cslm_joint_paper/
[10]https://cs.nyu.edu/ kcho/DMQA/
[11]For downloading and pre-processing, please refer to https://github.com/abisee/cnn-dailymail
[12]https://summari.es/
[13]https://duc.nist.gov/data.html
[14]http://opennmt.net/Models/

for question answering and is one of the earliest data sets that are available for tackling this problem. However, recently, two large-scale data sets that are solely designed for this problem were released. Stanford Question Answering data set (SQuAD)[15] (1.0 and 2.0) [115], [116] is a data set for reading comprehension and contains more than 100k pairs of questions and answers collected by crowdsourcing over a set of Wikipedia articles. The answer to each question is a segment that identifies the start and end indices of the answer within the article. The second data set is called TriviaQA[16] [117], and similar to SQuAD, it is designed for reading comprehension and question answering task. This data set contains 650k triples of questions, answers, and evidences (which helps to find the answer).

5) *Dialogue Generation:* The data set for this problem usually comprises of dialogues between different people. The OpenSubtitles data set[17] [118], Movie Dialog data set[18] [119], and Cornell Movie Dialogues Corpus[19] [120] are three examples of these types of data sets. OpenSubtitles contains conversations between movie characters for more than 20k movies in 20 languages. The Cornell Movie Dialogues corpus contains more than 220k dialogues between more than 10k movie characters.
6) *Semantic Parsing:* Recently, Zhong *et al.* [71] released a data set called WikiSQL[20] for this problem that contains 80 654 hand-annotated questions and SQL queries distributed across 24 241 tables from Wikipedia. Although this is not the only data set for this problem, it offers a larger set of examples from other data sets, such as WikiTableQuestion[21] [121] and Overnight [122].
7) *Sentiment Analysis:* For this application, Amazon product review[22] [123] data set is one of the largest data set that contains more than 82 million product reviews from May 1996 to July 2014 in its de-duplicated version. Another big data set for this task is the Stanford Sentiment Treebank (SSTb) [23] [95], which includes fine-grained sentiment labels for 215 154 phrases in the parse trees of 11 855 sentences.
8) *Natural Language Inference:* Stanford natural language inference (SNLI)[24] [124] is the standard data set for this task that contains 570k human-written English sentence pairs manually labeled for the three classes entailment, contradiction, and neutral. The multi-genre natural language inference (MultiNLI)[25] [125] corpus is another new data set that is collected through crowdsourcing and contains 433k sentence pairs annotated with textual entailment information.
9) *Semantic Role Labeling:* Proposition Bank (PropBank)[26] [126] is the standard data set for this task that contains a corpus of text annotation with information about basic semantic propositions in seven different languages.
10) *Relation Extraction:* Freebase[27] [127] is a huge data set containing billions of triples: the entity pair and the specific relationship between them that are selected from the New York Times corpus (NYT).
11) *Pronoun Resolution:* The OntoNotes 5.0 data set[28] is the standard data set for this task. Specifically, researchers use the Chinese portion of this data set to do the pronoun resolution in Chinese [106], [107], [109].
12) *Image Captioning:* There are two data sets that are mainly used in image captioning. The first one is the COCO data set[29] [128] that is designed for object detection, segmentation, and image captioning. This data set contains around 330k images, among which 82k images are used for training and 40k used for validation in image captioning. Each image has five ground-truth captions. SBU [129] is another data set that consists of 1M images from Flickr and contains descriptions provided by image owners when they uploaded the images to Flickr.
13) *Video Captioning:* For this problem, MSR-VTT[30] [130] and YouTube2Text/MSVD[31] [131] are two of the widely used data sets. MSR-VTT consists 10k videos from a commercial video search engine each containing 20 human annotated captions and YouTube2Text/MSVD that has 1970 videos each containing on an average 40 human annotated captions.
14) *Image Classification:* The most popular data set in computer vision is the MNIST data set[32] [132]. This data set consists of handwritten digits and contains a training set of 60k examples and a test set of 10k examples. Aside from this data set, there is a huge list of data sets that are used for various computer vision problems and explaining each of them is beyond the scope of this paper.[33]
15) *Speech Recognition:* LibriSpeech ASR Corpus[34] [133] is one of the main data sets used for the speech recognition task. This data set is free and composed of 1000 h of segmented and aligned 16-kHz English speech that is derived from audiobooks. The Wall Street Journal (WSJ) also has two Continuous Speech Recognition corpora containing 70 h of speech and text from a corpus of the WSJ news text. However, unlike the LibriSpeech data set, this data set is not

[15] https://rajpurkar.github.io/SQuAD-explorer/
[16] http://nlp.cs.washington.edu/triviaqa/
[17] http://opus.nlpl.eu/OpenSubtitles.php
[18] http://fb.ai/babi
[19] http://www.cs.cornell.edu/ cristian/Cornell_Movie-Dialogs_Corpus.html
[20] https://github.com/salesforce/WikiSQL
[21] https://nlp.stanford.edu/software/sempre/wikitable/
[22] http://jmcauley.ucsd.edu/data/amazon/
[23] https://nlp.stanford.edu/sentiment/
[24] https://nlp.stanford.edu/projects/snli/
[25] https://www.nyu.edu/projects/bowman/multinli/
[26] http://propbank.github.io/
[27] https://old.datahub.io/dataset/freebase
[28] https://catalog.ldc.upenn.edu/LDC2013T19
[29] http://cocodataset.org/
[30] http://ms-multimedia-challenge.com/2017/challenge
[31] http://www.cs.utexas.edu/users/ml/clamp/videoDescription/
[32] http://yann.lecun.com/exdb/mnist/
[33] Please refer to this link for a comprehensive list of data sets that are used in computer vision: http://riemenschneider.hayko.at/vision/dataset/
[34] http://www.openslr.org/12/

freely available, and researchers have to buy a license to use it. Similar to the WSJ data set, TIMIT[35] is another data set containing the read speech data. It contains time-aligned orthographic, phonetic, and word transcriptions of recordings for 630 speakers of eight major dialects of American English, in which each of them are reading ten phonetically sentences.

III. Reinforcement Learning Methods

In RL, the goal of an agent interacting with an environment is to maximize the expectation of the reward that it receives from the actions. Therefore, the focus is on maximizing one of the following objectives:

$$\max \ \mathbb{E}_{\hat{y}_1,\ldots,\hat{y}_T\sim\pi_\theta(\hat{y}_1,\ldots,\hat{y}_T)}[r(\hat{y}_1,\ldots,\hat{y}_T)] \tag{9}$$

$$\max_y \ A_\pi(s_t, y_t) \tag{10}$$

$$\max_y \ A_\pi(s_t, y_t) \rightarrow \text{Max}_y \ Q_\pi(s_t, y_t). \tag{11}$$

There are various ways, in which one can solve this problem. In this section, we explain the solutions in detail and provide their strengths and weaknesses. Different methods aim for solving this problem by trying one of the following approaches: 1) solve this problem through (9); 2) solve the expected discounted reward $\mathbb{E}[R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau]$; 3) solve it by maximizing the advantage function [see (10)]; and 4) solve it by maximizing Q-function using (11). Most of these methods are suitable choices for improving the performance of seq2seq models, but depending on the approach that is chosen for training the reinforced model, the training procedure for seq2seq model also changes. The first and one of the simplest algorithms that will be discussed in this section is the policy gradient (PG) method that aims to solve (9). Section III-B discusses the actor–critic (AC) methods that improve the performance of PG models by solving (10) through (7) expansion on Q-function. Section III-C discusses Q-learning models that aim at maximizing the Q-function [see (11)] to improve the PG and AC models. Finally, Section III-D will provide more details about some of the recent models that improve the performance of Q-learning models.

A. Policy Gradient

In all reinforcement algorithms, an agent takes some action according to a specific policy π. The definition of a policy varies according to the specific application that is being considered. For instance, in text summarization, the policy is a language model $p(y|X)$ that, given input X, tries to generate the output y. Now, let us assume that our agent is represented by an RNN and takes actions from a policy π_θ.[36] In a deterministic environment, where the agent takes discrete actions, the output layer of the RNN is usually a softmax function, and it generates the output from this layer. In Teacher Forcing, a set of ground-truth sequences are given, the actions are chosen according to the current policy during training, and the reward is observed only at the end of the sequence or when an end-of-sequence (EOS) signal is seen. Once the agent reaches the end of sequence, it compares the sequence of actions from the current policy ($\hat{y}_t$) against the ground-truth action sequence (y_t) and calculate a reward based on any specific evaluation metric. The goal of the training is to find the parameters of the agent in order to maximize the expected reward. This loss is defined as the negative expected reward of the full sequence

$$\mathcal{L}_\theta = -\mathbb{E}_{\hat{y}_1,\ldots,\hat{y}_T\sim\pi_\theta(\hat{y}_1,\ldots,\hat{y}_T)}[r(\hat{y}_1,\ldots,\hat{y}_T)] \tag{12}$$

where $\hat{y}_t$ is the action chosen by the model at time t and $r(\hat{y}_1,\ldots,\hat{y}_T)$ is the reward associated with the actions $\hat{y}_1,\ldots,\hat{y}_T$. Usually, in practice, one will approximate this expectation with a single sample from the distribution of actions acquired by the RNN model. Hence, the derivative for the above-mentioned loss function is given as follows:

$$\nabla_\theta \mathcal{L}_\theta = -\mathop{\mathbb{E}}_{\hat{y}_{1\cdots T}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\hat{y}_{1\cdots T}) r(\hat{y}_{1\cdots T})]. \tag{13}$$

Using the chain rule, (13) can be re-written as follows [134]:

$$\nabla_\theta \mathcal{L}_\theta = \frac{\partial \mathcal{L}_\theta}{\partial \theta} = \sum_t \frac{\partial \mathcal{L}_\theta}{\partial o_t}\frac{\partial o_t}{\partial \theta} \tag{14}$$

where o_t is the input to the softmax function. The gradient of the loss $\mathcal{L}_\theta$ with respect to o_t is given by [41], [134]

$$\frac{\partial \mathcal{L}_\theta}{\partial o_t} = (\pi_\theta(y_t|\hat{y}_{t-1}, s_t, c_{t-1}) - \mathbf{1}(\hat{y}_t))(r(\hat{y}_1,\ldots,\hat{y}_T) - r_b) \tag{15}$$

where $\mathbf{1}(\hat{y}_t)$ is the one-of-$|\mathcal{A}|$ representation of the ground-truth output and r_b is a baseline reward and could be any value as long, as it is not dependent on the parameters of the RNN network. Equation (15) is very similar to the gradient of a multi-class logistic regression. In logistic regression, the CE gradient is the difference between the prediction and the actual one-of-$|\mathcal{A}|$ representation of the ground-truth output

$$\frac{\partial \mathcal{L}_\theta^{CE}}{\partial o_t} = \pi_\theta(y_t|y_{t-1}, s_t, c_{t-1}) - \mathbf{1}(y_t). \tag{16}$$

Note that in (15), the generated output from the model is used as a surrogate ground truth for the output distribution, while in (16), the ground truth is used to calculate the gradient.

The goal of the baseline reward is to force the model to select actions that yield a reward $r > r_b$ and discourage those that have reward $r < r_b$. Since only one sample is being used to calculate the gradient of the loss function, it is shown that having this baseline would reduce the variance of the gradient estimator [41]. If the baseline is not dependent on the parameters of the model θ, (15) is an unbiased estimator. To prove this, we simply need to show that adding the baseline reward r_b does not have any effect on the expectation of loss

$$\begin{aligned}\mathop{\mathbb{E}}_{\hat{y}_{1\cdots T}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\hat{y}_{1\cdots T}) r_b] &= r_b \sum_{\hat{y}_{1\cdots T}} \nabla_\theta \pi_\theta(\hat{y}_{1\cdots T}) \\ &= r_b \nabla_\theta \sum_{\hat{y}_{1\cdots T}} \pi_\theta(\hat{y}_{1\cdots T}) = r_b \nabla_\theta 1 = 0.\end{aligned} \tag{17}$$

This algorithm is called REINFORCE [41] and is a simple yet elegant PG algorithm for seq2seq problems. One of the

[35] https://catalog.ldc.upenn.edu/ldc93s1

[36] In seq2seq model, this represents $\pi_\theta(y_t|\hat{y}_{t-1}, s_t, c_{t-1})$ in (1)

Algorithm 2 REINFORCE Algorithm

Input: Input sequences (X), ground-truth output sequences (Y),
and (preferably) a pre-trained policy (π_θ).
Output: Trained policy with REINFORCE.
Training Steps:
while not converged **do**
Select a batch of size N from X and Y.
Sample N full sequence of actions:
$\{\hat{y}_1, \ldots, \hat{y}_T \sim \pi_\theta(\hat{y}_1, \ldots, \hat{y}_T)\}_1^N$.
Observe the sequence reward and calculate the baseline r_b.
Calculate the loss according to Eq. (18).
Update the parameters of network $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_\theta$.
end while
Testing Steps:
for batch of input and output sequences X and Y **do**
Use the trained model and Eq. (4) to sample the output $\hat{Y}$.
Evaluate the model using a performance metric, e.g., $ROUGE_l$.
end for

challenges with this method is that the model suffers from high variance since only one sample is used for training at each time step. To alleviate this problem, at each training step, one can sample N sequences of actions and update the gradient by averaging over all these N sequences as follows:

$$\mathcal{L}_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t} \log \pi_\theta(\hat{y}_{i,t} | \hat{y}_{i,t-1}, s_{i,t}, c_{i,t-1}) \times (r(\hat{y}_{i,1}, \ldots, \hat{y}_{i,T}) - r_b). \quad (18)$$

Having this, the baseline reward could be set to the mean of the N rewards that are sampled, i.e., $r_b = 1/N \sum_{i=1}^{N} r(\hat{y}_{i,1}, \ldots, \hat{y}_{i,T})$. Algorithm 2 shows how this method works.

As another solution to reduce the variance of the model, self-critic (SC) models are proposed [40]. In these SC models, rather than estimating the baseline using current samples, the output of the model obtained by a greedy search (the output at the time of inference) is used as the baseline. Hence, the sampled output of the model is used as $\hat{y}_t$, and the greedy selection of the final output distribution is used for $\hat{y}_t^g$, where the superscript g indicates greedy selection. Following this mechanism, the new objective for the REINFORCE model would become as follows:

$$\mathcal{L}_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t} \log \pi_\theta(\hat{y}_{i,t} | \hat{y}_{i,t-1}, s_{i,t}, c_{i,t-1}) \times \left(r(\hat{y}_{i,1}, \ldots, \hat{y}_{i,T}) - r\left(\hat{y}_{i,1}^g, \ldots, \hat{y}_{i,T}^g\right)\right). \quad (19)$$

Fig. 2 shows how an attention-based pointer-generator seq2seq model can be used to extract the reward and its baseline in the SC model.

The second problem with this method is that the reward is only observed after the full sequence of actions is sampled. This might not be a pleasing feature for most of the seq2seq models. If we see the partial reward of a given action at time t, and the reward is bad, the model needs to select a better action for the future to maximize the reward. However, in the REINFORCE algorithm, the model is forced to wait untill the end of the sequence to observe its performance. Therefore, the model often generates poor results or takes longer to converge. This problem magnifies especially in the beginning of the training phase, where the model is initialized randomly and, thus, selects arbitrary actions. To alleviate this problem to a certain extent, Ranzato *et al.* [28] suggested to pre-train the model for a few epochs using the CE loss and then slowly switch to the REINFORCE loss. Finally, as another way to solve the high variance problem of the REINFORCE algorithm, importance sampling [135], [136] can also be used. The basic underlying idea of using the importance sampling with the REINFORCE algorithm is that rather than sampling sequences from the current model, one can sample them from an old model and use them to calculate the loss.

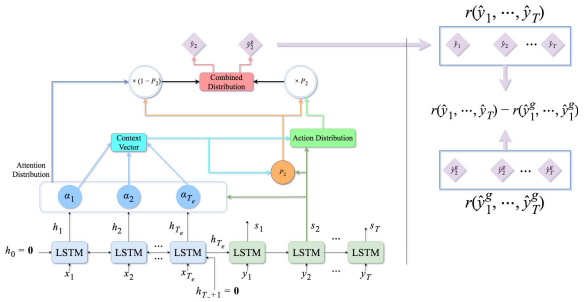

Fig. 2. Simple attention-based pointer-generation seq2seq model with SC reward. At each decoding step, the context vector for that decoder is calculated and combined with the decoder output to get the action distribution. In pointer-generation model, the attention distribution is further combined with the action distribution through switches called pointers to get the final distribution over the actions. From each output distribution, a specific action $\hat{y}_2$ is sampled, and the greedy action $\hat{y}_2^g$ is extracted. The difference of the rewards from sampling and greedy sequence is used to update the loss function.

B. Actor–Critic Model

As mentioned in Section III-A, adding a baseline reward is a necessary component of the PG algorithm in order to reduce the variance of the model. In PG, the average reward from multiple samples in the batch was used as the baseline reward for the model. In the AC model, the goal is to train an estimator for calculating the baseline reward. For computing this quantity, AC models try to maximize the advantage function through (7) extension. Therefore, these methods are also called advantage AC (A2C) models.

In these models, the goal is to solve this problem using the following objective:

$$A_\pi(s_t, y_t) = Q_\pi(s_t, y_t) - V_\pi(s_t) = r_t + \gamma \, \mathbb{E}_{s_{t'} \sim \pi(s_{t'}|s_t)}[V_\pi(s_{t'})] - V_\pi(s_t). \quad (20)$$

Similar to the PG algorithm, to avoid the expensive inner expectation calculation, we can only sample once and approximate advantage function as follows:

$$A_\pi(s_t, y_t) \approx r_t + \gamma \, V_\pi(s_{t'}) - V_\pi(s_t). \quad (21)$$

Now, in order to estimate $V_\pi(s)$, a function approximator can be used to approximate the value function. In AC, neural

networks are typically used as the function approximator for the value function. Therefore, we fit a neural network $V_\pi(s;\Psi)$ with parameter Ψ to approximate the value function. Now, if we consider $r_t + \gamma V_\pi(s_{t'})$ as the expectation of reward-to-go at time t, $V_\pi(s_t)$ could play as a surrogate for the baseline reward. Similar to the PG, the variance of the model would be high since only one sample is used to train the model. Therefore, the variance can be reduced using multiple samples. In the AC model, the actor (our policy, θ) provides samples (policy states at time t and $t+1$) for the critic [neural network estimating value function, $V_\pi(s;\Psi)$], and the critic returns the estimation to the actor, and finally, the actor uses these estimations to calculate the advantage approximation and update the loss according to the following equation:

$$\mathcal{L}_\theta = \frac{1}{N}\sum_{i=1}^{N}\sum_{t}\log \pi_\theta(\hat{y}_{i,}|\hat{y}_{i,t-1}, s_{i,t}, c_{i,t-1})A_\Psi(s_{i,t}, y_{i,t}). \tag{22}$$

Therefore, in the AC models, the inference at each time t would be as follows:

$$\arg\max_{y} \pi_\theta(\hat{y}_t|\hat{y}_{t-1}, s_t, c_{t-1})A_\Psi(s_t, y_t). \tag{23}$$

Fig. 3 provides an illustration of how this model works at one of the decoding steps.

1) Training Critic Model: As mentioned in Section III-B, the critic is a function estimator that tries to estimate the expected reward-to-go for the model at time t. Therefore, training the critic is basically a regression problem. Usually, in AC models, a neural network is used as the function approximator, and the value function is trained using the mean squared error (mse)

$$\mathcal{L}(\Psi) = \frac{1}{2}\sum_{i}||V_\Psi(s_i) - \upsilon_i||^2 \tag{24}$$

where $\upsilon_i = \sum_{t'=t}^{T} r(s_{i,t'}, y_{i,t'})$ is the true reward-to-go at time t. During training the actor model, we collect (s_i, υ_i) pairs and pass them to the critic model to train the estimator. This model is called on-policy AC , which refers to the fact that the samples are collected at the current time to train the critic model. However, the samples that are passed to the critic will be correlated to each other, which causes poor generalization for the estimator. These methods could be turned to off-policy by collecting training samples into a memory buffer and select mini-batches from this memory buffer and train the critic network. Off-policy AC provides better training due to avoiding the correlation of samples that exist in the on-policy methods. Therefore, most of the models that we discuss in this paper are primarily off-policy and use a memory buffer for training the critic model.

Algorithm 3 shows the batch AC algorithm since, for training the critic network, we use a batch of the state–rewards pair. In the online AC algorithm, the critic network is simply updated using just one sample, and, as expected, the online AC algorithm has a higher variance due to reliance on one sample for training the network. To alleviate this problem for online AC, we can use synchronous advantage AC learning

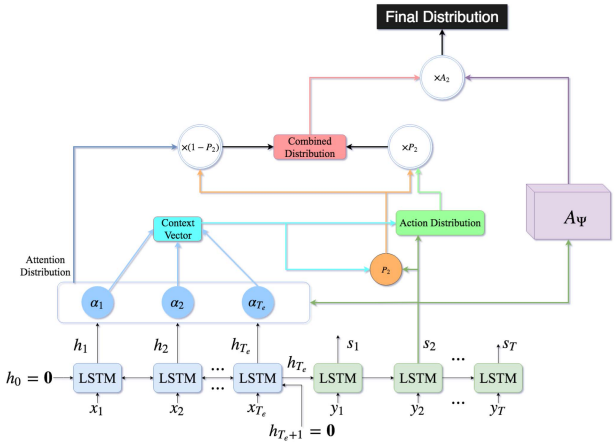

Fig. 3. Simple AC model with an attention-based pointer-generation seq2seq model as the actor. The critic model is shown on the right-hand side of the picture with a purple box. The purple box A_Ψ, which represents the critic model, takes as input the decoder output at time $t = 2$, i.e., s_2, and estimate the advantage values through either (value function estimation, DQN, DDQN, or dueling net) for each action.

or asynchronous advantage AC (A3C) learning [137]. In the synchronous approach, N different threads are used to train the model and each thread performs online AC for one sample, and at the end of the algorithm, the gradient of these N threads is used to update the gradient of the actor model. In the more widely used A3C algorithm, as soon as a thread calculates θ, it will send the update to other threads, and other threads use the updated θ to train the model. A3C is an on-policy method with multistep returns, while there are other methods, such as Retrace [138], UNREAL [139], and Reactor [140], that provide the off-policy variations of this model by using the memory buffer. Also, ACER [141] mixes on-policy (from the current run) and off-policy (from memory) to train the critic network.

In general, AC models usually have low variance due to the batch training and the use of critic as the baseline reward, but they are not unbiased if the critic is erroneous and makes a lot of mistakes. As mentioned in Section III-A, the PG algorithm has high variance, but it provides an unbiased estimator. Now, if the PG and AC models are combined, we will likely be ending up with a model that has no bias and low variance. This idea comes from the fact that for deterministic policies (such as seq2seq models), a partially observable loss could be driven by using the Q-function as follows [39], [142]:

$$\mathcal{L}_\theta = \frac{1}{N}\sum_{i=1}^{N}\sum_{t}\log \pi_\theta(\hat{y}_{i,t}|\hat{y}_{i,t-1}, s_{i,t}, c_{i,t-1}) \times (Q_\Psi(s_{i,t}) - V_{\Psi'}(s_{i,t})) \tag{25}$$

However, this model requires training two different networks for Q_Ψ function and $V_{\Psi'}$ function as the baseline. Note that the same model cannot be used to estimate both Q-function and value function since the estimator will not be an unbiased estimator anymore [143]. As yet another solution to create a trade-off between the bias and variance in AC, Schulman *et al.* [144] proposed the generalized advantage estimation (GAE) model

Algorithm 3 Batch AC Algorithm

Input: Input sequences (X), ground-truth output sequences (Y),
and (preferably) a pre-trained Actor model (π_θ).
Output: Trained Actor and Critic models.
Training Steps:
Initialize the Actor (Seq2seq) model, π_θ.
Initialize the Critic (ValueNet) model, V_Ψ.
while not converged **do**
  Training Actor:
  Select a batch of size N from X and Y.
  Sample N full sequences of actions based on the Actor. model, π_θ.
  for $n = 1, \ldots, N$ **do**
    for $t = 1, \ldots, T$ **do**
      Calculate the true (discounted) reward-to-go: $v_t = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, y_{i,t'})$.
      Store training pairs for Critic: (s_t, v_t).
    end for
  end for

  Training Critic:
  Select a batch of size N_c from the pool of state-rewards pairs.
  collected from Actor.
  for $n = 1, \ldots, N_c$ **do**
    Collect the value estimates $\hat{v}_n$ from V_Ψ for each state-rewards pair.
  end for
  Minimize the Critic loss using Eq. (24).

  Updating Actor:
  Use the estimated value for $V_\Psi(s_t)$ and $V_\Psi(s_{t'})$
  to calculate the loss using Eq. (22).
  Update parameters of the model using $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta)$.
end while

as follows:

$$A_\Psi^{\text{GAE}}(s_t, y_t) = \sum_{i=t}^{T} (\gamma\,\lambda)^{i-t} (r(s_i, y_i) + \gamma\, V_\Psi(s_{i+1}) - V_\Psi(s_i)) \tag{26}$$

where λ controls the tradeoff between the bias and variance, such that large values of λ yield to larger variance and lower bias, while small values of λ do the opposite.

C. Actor–Critic With Q-Learning

As mentioned in Section III-B, the value function is used to maximize the advantage function. As an alternative to solve the maximization of advantage estimates, we can try to solve the following objective function:

$$\text{Max}_y\ A_\pi(s_t, y_t) \rightarrow \text{Max}_y\ Q_\pi(s_t, y_t) - \underbrace{V_\pi(s_t)}_{0}. \tag{27}$$

This is true since we are trying to find the actions that maximize the advantage estimate, and since value function does not rely on the actions, we can simply remove them from the maximization objective. Therefore, the advantage maximization problem is simplified to Q-function estimation problem. This method is called Q-learning, and it is one of the most commonly used algorithms for RL problems. The Q-learning is called to be a family of the off-policy algorithm used to learn a Q-function. Similar to this method, the SARSA algorithm [145] is an on-policy algorithm for calculating the Q-function. The major difference between SARSA and Q-Learning is that the maximum reward for the next state is not necessarily used for updating the Q-values. In Q-learning, the critic tries to provide an estimation for the Q-function. Therefore, given that the policy π_θ is being used, our goal is to maximize the following loss at each training step:

$$\mathcal{L}_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t} \log \pi_\theta(\hat{y}_{i,t} | \hat{y}_{i,t-1}, s_{i,t}, c_{i,t-1}) Q_\Psi(s_{i,t}, y_{i,t}). \tag{28}$$

Similar to the value network training, the Q-function estimation is a regression problem, and the mse is used for training it. However, one of the differences between the Q-function training and the value function training is the way in which the true estimates are chosen. In value function estimation, the ground-truth data are used to calculate the true reward-to-go as $v_i = \sum_{t'=t}^{T} r(s_{i,t'}, y_{i,t'})$; however, in Q-learning, the estimation from the network approximator itself is used to train the regression model

$$\mathcal{L}(\Psi) = \frac{1}{2} \sum_{i} ||Q_\Psi(s_i, y_i) - q_i||^2$$
$$q_i = r_t + \gamma \max_{y'} Q_\Psi(s_i', y_i') \tag{29}$$

where s_i' and y_i' are the state and action at the next time, respectively. Although the Q-value estimation has no direct relation to the true Q-values calculated using ground-truth data, in practice, it is known to provide good estimation and faster training due to not collecting ground-truth reward at each step of the training. However, there are no rigorous studies that analyze how far these estimates are from the true Q-values. As shown in (29), the true Q estimations are calculated using the estimation from network approximator at time $t + 1$, i.e., $\max_y' Q_\Psi(s_i', y_i')$. Although not relying on the true ground-truth estimation and explicitly using the reward function might seem to be a bad idea, however, in practice, it is shown that these models provide better and more robust estimators. Therefore, the training process in Q-learning consists of first collecting a data set of experiences $e_t = (s_t, y_t, s_{t'}, r_t)$ during training our actor model and then use them to train the network approximator. This is the standard way of training the Q-network and was frequently used in earlier temporal-difference learning models. However, there is a problem with this method. Generally, the AC models with a neural network as function estimator are tricky to train, and unless there are guarantees that the estimator is good, the model does not converge. Although the original Q-learning method is proven to converge [146], [147], when a neural

network is used to approximate the estimator, the convergence guarantee no longer holds. Usually, since samples are coming from specific sets of sequences, there is a correlation between the samples that are chosen to train the model. Thus, this may cause any small updates to Q-network to significantly change the data distribution and ultimately affects the correlations between Q and the target values. Recently, Mnih *et al.* [45] proposed using an *experience buffer* [148][37] to store the experiences from different sequences and then randomly select a batch from this data set and train the Q-network. Similar to the off-policy AC model, one benefit of using this buffer is the potential to increase the efficiency of the model by re-using the experiences in multiple updates and reducing the variance of the model since by sampling uniformly from the buffer, the correlation of samples used in the updates is reduced. As another improvement to the experience buffer, a prioritized version of this buffer is used, in which to select the mini-batches during training, only samples that have low temporal difference error are selected [149]. Algorithm 4 provides the pseudocode for a Q-learning algorithm called deep Q-network (DQN).

D. Advanced Q-Learning

1) Double Q-Learning: One of the problems with the DQN is the overestimation of Q-values, as discussed in [150] and [151]. Specifically, the problem lies in the fact that the ground-truth reward is not used to train these models, and the same network is used to calculate both the estimation of network $Q_\Psi(s_i, y_i)$ and true values for regression training, q_i. To alleviate this problem, one could use two different networks, in which the first one chooses the best action when calculating $\max_{y'} Q_\Psi(s'_n, y'_n)$ and the other calculates the estimation of Q value, i.e., $Q_\Psi(s_i, y_i)$. In practice, a modified version of the current DQN network is used as the second network, in which the current network freezes its parameters for a certain period of time and updates the second network periodically. Let us call the second network as the target network with parameter Ψ'. We know that $\max_{y'} Q_\Psi(s'_n, y'_n)$ is the same as choosing the best action according to the network Q_Ψ. Therefore, this equation can be re-written as $Q_\Psi(s'_t, \arg\max_{y'_t} Q_\Psi(s'_t, y'_t))$. As shown in this equation, Q_Ψ is used for both calculating the Q-value and finding the best action. Given a target network, the best action is chosen using our target network, and the Q-value is estimated using the current network. Hence, using the target network, $Q_{\Psi'}$, the Q-estimation will be given as follows:

$$q_t = \begin{cases} r_t & s'_n == \text{EOS} \\ r_t + \gamma\, Q_\Psi(s'_t, \arg\max_{y'_t} Q_{\Psi'}(s'_t, y'_t)) & \text{otherwise} \end{cases} \tag{30}$$

where EOS stands for the end-of-sequence action. This method is called double DQN (DDQN) [150], [152] and is shown to resolve the problem of overestimation in DQN and provides more realistic estimations. However, even this model suffers from the fact that there is no relation between the true Q-values and the estimation provided by the network. Algorithm 5 shows the pseudocode for this model.

[37] In some studies, it is called a *replay buffer*.

Algorithm 4 Deep Q-Learning

Input: Input sequences (X), ground-truth output sequences (Y),
and preferably a pre-trained Actor model (π_θ).
Output: Trained Actor and Critic models.
Training Steps:
Initialize the Actor (Seq2seq) model, π_θ.
Initialize the Critic (Q-Net) model, Q_Ψ.
while not converged **do**
  Training Seq2seq Model:
  Select a batch of size N from X and Y.
  Sample N full sequences of actions based on the Actor model, π_θ.
  for $n = 1, \ldots, N$ **do**
    for $t = 1, \ldots, T$ **do**
      Collect experience $e_t = (s_t, y_t, s_{t'}, r_t)$ and add them to the *experience buffer*.
    end for
  end for

  Training Q-Net:
  Select a batch of size N_q from the *experience buffer*. based on the reward.
  for $n = 1, \ldots, N_q$ **do**
    Estimate $\hat{q_n} = Q_\Psi(s_n, y_n)$.
    Calculate the true estimation:
    $q_n = \begin{cases} r_n & s'_n\text{==EOS} \\ r_n + \gamma\, max_{y'} Q_\Psi(s'_n, y'_n) & \text{otherwise.} \end{cases}$
    Store $(\hat{q}_n, q_n)$.
  end for
  Updating Q-Net:
  Minimize the loss using Eq. (29).
  Update the parameters of network, Ψ.

  Updating Seq2seq Model:
  Use the estimated Q values for $\hat{q}_n = Q_\Psi(s_n, y_n)$.
  to calculate the loss using Eq. (28).
  Update parameters of the model using $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta)$.
end while

2) Dueling Networks: DDQN tried to solve one of the problems with DQN model by using two networks, in which the target network selects the next best action, while the current network estimates the Q-values, given the action selected by the target. However, in most applications, it is unnecessary to estimate the value of each action choice. This is especially important in discrete problems with a large set of possible actions, where only a small portion of actions are suitable. For instance, in text summarization, the output of the model is a vector of the distribution over the vocabulary, and therefore, the output has the same dimension as the vocabulary size that is usually selected to be between 50k and 150k. In most of the applications that use DDQN, the action space is limited to less than a few hundred. For instance, in an Atari game, the possible actions could be to move left, right, up, down, and shoot. Therefore, using DDQN would be easy for these types of applications. Recently, Wang *et al.* [153] proposed the idea of using a dueling net to overcome this problem.

Algorithm 5 Double Deep Q-Learning

Input: Input sequences (X), ground-truth output sequences (Y),
and preferably a pre-trained Actor model (π_θ).
Output: Trained Actor and Critic models.
Training Steps:
Initialize the Actor (Seq2seq) model, π_θ.
Initialize the two Critic models:
current Q-Net, Q_Ψ, and target Q-net, $Q_{\Psi'}$: $Q_{\Psi'} \leftarrow Q_\Psi$.
while not converged **do**
  Training Seq2seq Model:
  Select a batch of size N from X and Y.
  Sample N full sequences of actions based on the Actor model, π_θ.
  for $n = 1, \ldots, N$ **do**
    for $t = 1, \ldots, T$ **do**
      Collect experience $e_t = (s_t, y_t, s_{t'}, r_t)$ and add them to the *experience buffer*.
    end for
  end for

  Training Q-Net:
  Select a batch of size N_q from the *experience buffer* based on the reward.
  for $n = 1, \ldots, N_q$ **do**
    Estimate $\hat{q}_n = Q_\Psi(s_n, y_n)$
    Calculate the true estimation:

$$q_n = \begin{cases} r_n & s'_n == EOS \\ r_n + \gamma\, Q_\Psi(s'_n, \\ \quad \arg\max_{y'_t} Q_{\Psi'}(s'_t, y'_t)) & otherwise. \end{cases}$$

    Store $(\hat{q}_n, q_n)$.
  end for
  Updating current Q-Net:
  Minimize the loss using Eq. (29).
  Update the parameters of network, Ψ.

  Updating target Q-Net every N_u iterations:
  $\Psi' \leftarrow \Psi$ or using Polyak averaging:
  $\Psi' \leftarrow \tau\Psi' + (1-\tau)\Psi$, $\tau = \frac{1000 - (\text{Current Step}\%1000)}{1000}$.

  Updating Seq2seq Model:
  Use the estimated Q-values for $\hat{q}_n = Q_\Psi(s_n, y_n)$ to calculate the loss using Eq. (28).
  Update parameters of the model using $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta)$.
end while

In their proposed method, rather than estimating the Q-values directly from the Q-net, two different values are estimated for the value function and advantage function as follows:

$$Q_\Psi(s_t, y_t) = V_\Psi(s_t) + A_\Psi(s_t, y_t). \tag{31}$$

In order to be able to calculate $V_\Psi(s_t)$, the value estimates are replicated $|\mathcal{A}|$ times. However, as discussed in [153], using (33) to calculate Q is bad and can potentially yield poor performance since (33) is unidentifiable in the sense that a constant can be added to $V_\Psi(s_t)$ and subtracted the same constant from $A_\Psi(s_t, y_t)$. To solve this problem, the authors suggested forcing the advantage estimator to have a zero at the selected action

$$Q_\Psi(s_t, y_t) = V_\Psi(s_t) + (A_\Psi(s_t, y_t) - \max_y A_\Psi(s_t, y)). \tag{32}$$

This way, for the action $y^* = \arg\max_y Q_\Psi(s_t, y) = \arg\max_y A_\Psi(s_t, y)$, $Q_\Psi(s_t, y^*) = V_\Psi(s_t)$ is obtained. As an alternative to (32) and to make the model more stable, the author suggested to replace the max operator with average

$$Q_\Psi(s_t, y_t) = V_\Psi(s_t) + \left(A_\Psi(s_t, y_t) - \frac{1}{|\mathcal{A}|} \sum_y A_\Psi(s_t, y) \right). \tag{33}$$

Note that the dueling net will not decrease the number of actions but will provide a better normalization over the target distribution. Similar to DQN and DDQN, this model also suffers from the fact that there is no relation between the true values of Q-function and the estimation provided by the network. In Section V, we propose a simple and effective solution to overcome this problem by doing scheduled sampling between the Q-value estimations and true Q-values to pre-train our function approximator. Fig. 4 summarizes some of the strengths and weaknesses of these different RL methods.

IV. Combining RL With seq2seq Models

In this section, we will provide some of the recent models that combined the seq2seq training with RL techniques. For most of these models, the main goal is to solve the train/test evaluation mismatch problem that exists in all previously described seq2seq models. This is usually done by adding a reward function to the training objective. There are a growing number of research works that used the REINFORCE algorithm to improve the current state-of-the-art seq2seq models. However, more advanced techniques, such as AC models, DQN, and DDQN, have not been used often for these tasks. As mentioned earlier, one of the main difficulties of using Q-Learning and its derivatives is the large action space for seq2seq models. For instance, in text summarization, the model should provide estimates for each word in the vocabulary, and therefore, the estimation could be inferior even with a well-trained model. Due to these reasons, researchers mostly focused on the easier yet problematic approaches, such as REINFORCE algorithm to train the seq2seq model. Therefore, combining the power of Q-Learning training with seq2seq model is still considered to be an open area of research. Table III shows the policy, action, and reward function for each seq2seq task, and Table IV summarizes these models along with the respective seq2seq application and specific RL algorithm they used to improve that application.

A. Policy Gradient and REINFORCE Algorithm

As mentioned in Section III-A, in PG, the reward of the sampled sequence is observed at the end of the sequence generation and backpropagate that error equally to all the decoding steps according to (15). Also, we talked about the exposure bias problem that exists in seq2seq models during training the decoder because of using the CE error. The idea of improving generation by letting the model use its own predictions at

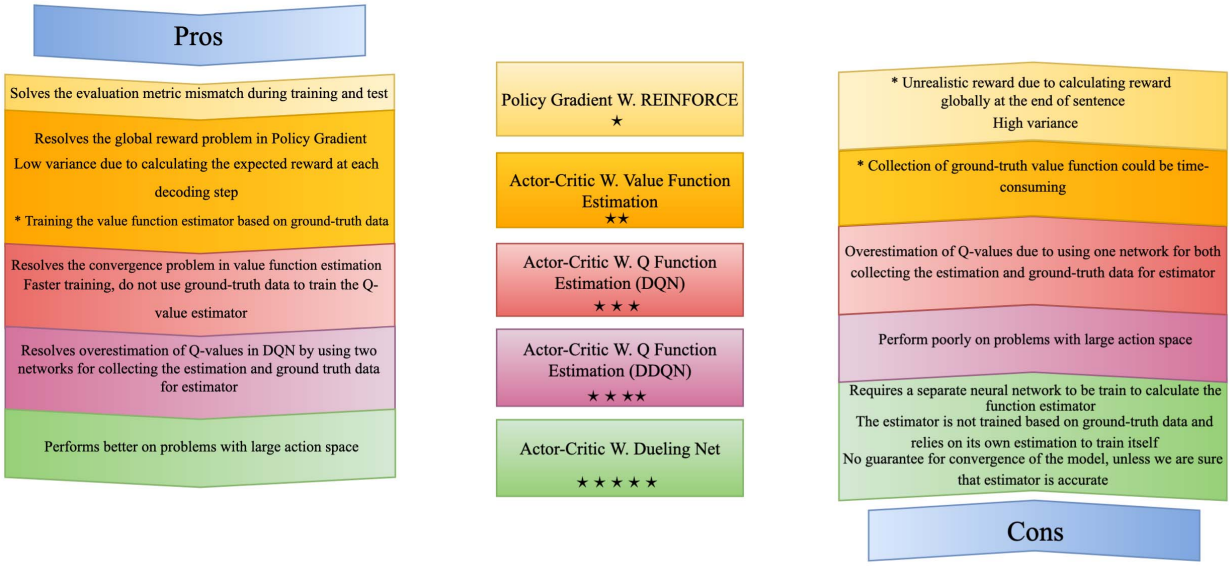

Fig. 4. List of advantages and drawbacks of different RL models. The advantages are listed, such that each method covers all the strengths of its previous methods, and the drawbacks are listed, such that each method has all the weaknesses of the previous ones. For instance, AC w. Dueling Net has all the pros of the previous models listed above it, and AC w. Value Function Estimation suffers from all the cons of the methods listed below it. The features that are also model-dependent are shown with "$*$" and those features do not exist in any other model. Each "$\star$" shows how hard it is to implement these models in a real-world application.

TABLE III

POLICY, ACTION, AND REWARD FUNCTION FOR DIFFERENT SEQ2SEQ TASKS

Seq2seq Task	Policy	Action	Reward
Text Summarization Headline Generation Machine Translation Question Generation	Attention-based models, pointer-generators, etc.	Selecting the next token for summary, headline, and translation	ROUGE, BLEU
Question Answering	Seq2seq model	Selecting the answer from a vocabulary or selecting the start and end index of the answer in the input document	F1 Score
Image Captioning Video Captioning	Seq2seq model	Selecting the next token for the caption	CIDEr, SPICE, METEOR
Speech Recognition	Seq2seq model	Selecting the next token for the speech	Connectionist Temporal Classification (CTC)
Dialog Generation	Seq2seq model	Dialogue utterance to generate	BLEU Length of dialogue Diversity of dialogue

training time was first proposed by Daumé *et al.* [154]. Based on their proposed method, SEARN, the structured prediction problems can be cast as a particular instance of RL. The basic idea is to let the model use its own predictions at training time to produce a sequence of actions (e.g., the choice of the next word). Then, a greedy search algorithm is run to determine the optimal action at each time step, and the policy is trained to predict that action. An imitation learning framework was proposed by Ross *et al.* [155] in a method called DAGGER, where an oracle of the target word, given the current predicted word, is required. However, for tasks such as text summarization, computing the oracle is infeasible due to the large action space. This problem was later addressed by the "Data As Demonstrator" (DAD) model [156], where the target action at step k is the kth action taken by the optimal policy. One drawback of DAD is that at every time step, the target label is always selected from the ground-truth data, and if the generated summaries are shorter than the ground-truth summaries, the model still forces to generate outputs that could already exist in the model. One way to avoid this problem in DAD is to use a method called End2EndBackProp [28], in which, at each step t, the top-k actions are retrieved from the model, the normalized probabilities of these actions are used as weights (of importance), and the normalized combination of their representation is fed to the next decoding step.

Finally, the REINFORCE algorithm [41] tries to overcome all these problems by using the PG rewarding function and avoiding the CE loss by using the sampled sequence as the ground truth to train the seq2seq model [see (18)]. In real-world applications, the training is usually started with the CE loss, and a pre-trained model is acquired. Then, the REINFORCE algorithm is used to train the model. Some of the earliest adoptions of REINFORCE algorithm for training seq2seq models are in computer vision [90], [157], image captioning [19], and speech recognition [86]. Recently, other researchers showed that using a combination of CE loss and REINFORCE loss could yield a better result than just simply performing the pre-training. In these models, training is started by using the CE loss and is slowly switched from CE loss to REINFORCE loss to train the model. There are

TABLE IV
SUMMARY OF SEQ2SEQ APPLICATIONS THAT USED VARIOUS RL METHODS

Reference	Suffers From Exposure Bias	Mismatch on Train/Test Measure	Observe Full Reward	RL Algorithm	Seq2seq Application
Policy Gradient Based Models					
SEARN [154]	No	Yes	No Reward	PG	Sequence Labeling Syntactic Chunking
DAD [156]	No	Yes	No Reward	PG	Time-Series Modeling
Qin *et al.* [105]	No	Yes	Yes	PG	Relation Extraction
Yin *et al.* [109]	No	Yes	Yes	PG	Pronoun Resolution
MIXER [28]	No	No	Yes	PG w. REINFORCE	Machine Translation Text Summarization Image Captioning
Wu *et al.* [169]	No	No	Yes	PG w. REINFORCE	Text Summarization
Narayan *et al.* [159]	No	No	Yes	PG w. REINFORCE	Text Summarization
Kreutzer *et al.* [170]	No	No	Yes	PG w. REINFORCE	Machine Translation w. Human Bandit Feedback
Pan *et al.* [98]	No	No	Yes	PG w. REINFORCE	Natural Language Inference
Liang *et al.* [171]	No	No	Yes	PG w. REINFORCE	Semantic Parsing
Li *et al.* [163]	No	No	Yes	PG w. REINFORCE	Dialogue Generation
Yuan *et al.* [62]	No	No	Yes	PG w. REINFORCE	Question Generation
Mnih *et al.* [157]	Yes	No	Yes	PG w. REINFORCE	Computer Vision
Ba *et al.* [90]	Yes	No	Yes	PG w. REINFORCE	Computer Vision
Xu *et al.* [19]	Yes	No	Yes	PG w. REINFORCE	Image Captioning
Self-Critic Models with REINFORCE Algorithm					
Rennie *et al.* [40]	Yes	No	Yes	SC w. REINFORCE	Image Captioning
Paulus *et al.* [13]	No	No	Yes	SC w. REINFORCE	Text Summarization
Wang *et al.* [158]	No	No	Yes	SC w. REINFORCE	Text Summarization
Pasunuru *et al.* [161]	No	No	Yes	SC w. REINFORCE	Video Captioning
Yeung *et al.* [172]	No	No	Yes	SC w. REINFORCE	Action Detection in Video
Zhou *et al.* [162]	No	No	Yes	SC w. REINFORCE	Speech Recognition
Hu *et al.* [164]	No	No	Yes	SC w. REINFORCE	Question Answering
Actor-Critic Models with Policy Gradient and Q-Learning					
He *et al.* [165]	Yes	No	No	AC	Machine Translation
Li *et al.* [168]	Yes	No	No	AC	Machine Translation Text Summarization
Bahdanau *et al.* [39]	Yes	No	No	PG w. AC	Machine Translation
Li *et al.* [46]	Yes	No	No	PG w. AC	Text Summarization
Chen *et al.* [55]	Yes	No	Yes	PG w. AC	Text Summarization
Keneshloo *et al.* [173]	Yes	No	No	PG w. AC	Text Summarization
Zhang *et al.* [166]	Yes	No	No	PG w. AC	Image Captioning
Liu *et al.* [136]	Yes	No	No	PG w. AC	Image Captioning
DARLA [174]	Yes	No	No	AC	Domain Adaptation

various ways in which one can do the transition from CE loss to REINFORCE loss. Ranzato *et al.* [28] used an incremental scheduling algorithm called "MIXER" that combines DAGGER [155] with DAD [156] methods. In this method, the RNN is trained with the CE loss for N_{CE} epochs using the ground-truth sequences. This ensures that the model starts off with a much better policy than a random one because, now, the model can focus on promising regions of the search space. Then, they use an annealing schedule in order to gradually teach the model to produce stable sequences. Therefore, after the initial N_{CE} epochs, they continue training the model for $N_{\text{CE}} + N_R$ epochs, such that for every sequence, they use $\mathcal{L}_{\text{CE}}$ for the first $(T - \delta)$ steps and the REINFORCE algorithm for the remaining δ steps. The MIXER model was successfully used in a variety of tasks, such as text summarization, image captioning, and machine translation.

Another way to handle the transition from using CE loss to REINFORCE loss is to use the following combined loss:

$$\mathcal{L}_{\text{mixed}} = \eta \mathcal{L}_{\text{REINFORCE}} + (1 - \eta)\mathcal{L}_{\text{CE}} \tag{34}$$

where $\eta \in (0, 1)$ is the parameter that controls the transition from CE to REINFORCE loss. In the beginning of the training, $\eta = 0$ and the model completely relies on CE loss, while as the training progresses, the η value is increased in order to slowly reduce the effect of CE loss. By the end of the training process (where $\eta = 1$), the model completely uses the REINFORCE loss for training. This mixed training loss was used in many of the recent works on text summarization [13], [46], [158], [159], paraphrase generation [160], image captioning [40], video captioning [161], speech recognition [162], dialogue generation [163], question answering [164], and question generation [62].

B. Actor–Critic Models

One of the problems with the PG model is that we need to sample the full sequences of actions and observe the reward at the end of the generation. This, in general, will be problematic since the error of generation accumulates over time, and usually, for long sequences of actions, the final sequence is so far away from the ground-truth sequence. Thus, the reward of the final sequence would be small, and the model would take a lot of time to converge. To avoid this problem, AC models observe the reward at each decoding step using the critic model and fix the sequence of future

actions that the actor is going to take. The critic model usually tries to maximize the advantage function through the estimation of value function or Q-function. As one of the early attempts of using AC models, Bahdanau *et al.* [39] and He *et al.* [165] used this model for the problem of machine translation. Bahdanau *et al.* [39] used temporal-difference learning for advantage function estimation by considering the Q-value for the next action, i.e., $Q(s_t, y_{t+1})$, as a surrogate for its true value at time t, i.e., $V_\Psi(s_t)$. We mentioned that for a deterministic policy, $y^* = \arg\max_y Q(s, y)$, it follows that $Q(s, y^*) = V(s)$. Therefore, the Q-value for the next action could be used as the true estimates of the value function at the current time. To accommodate for the large action space, they also use the shrinking estimation trick that was used in dueling net to push the estimate to be closer to their means. In addition, the critic training is done through the following mixed objective function:

$$\mathcal{L}(\Psi) = \frac{1}{2}\sum_i ||Q_\Psi(s_i, y_i) - q_i||^2 + \eta \bar{Q}_i$$
$$\bar{Q}_i = \sum_y \left(Q_\Psi(y, s_i) - \frac{1}{|\mathcal{A}|}\sum_{y'} Q_\Psi(y', s_i) \right) \quad (35)$$

where q_i is the true estimation of Q from a delayed actor. The idea of using delayed actor is similar to the idea used in double Q-learning, where a delayed target network is used to get the estimation of the best action. Later, Zhang *et al.* [166] used a similar model on the image captioning task.

He *et al.* [165] proposed a value network that uses a semantic matching and a context-coverage module and passed them through a dense layer to estimate the value function. However, their model requires a fully trained seq2seq model to train the value network. Once the value network is trained, they use the trained seq2seq model and trained value estimation model to do the beam search during translation. Therefore, the value network is not used during the training of the seq2seq model. During inference, however, similar to the AlphaGo model [167], rather than multiplying the advantage estimates (value or Q estimates) to the policy probabilities [such as in (23)], they combine the output of the seq2seq model and the value network as follows:

$$\eta \times \frac{1}{T}\log \pi(\hat{y}_{1\cdots T}|X) + (1-\eta) \times \log V_\Psi(\hat{y}_{1\cdots T}) \quad (36)$$

where $V_\Psi(\hat{y}_{1\cdots T})$ is the output of the value network and η controls the effect of each score.

In a different model, Li *et al.* [168] proposed a model that controls the length of seq2seq model using RL-based ideas. They train a Q-value function approximator that estimates the future outcome of taking an action y_t in the present and then incorporate it into a score $S(y_t)$ at each decoding step as follows:

$$S(y_t) = \log \pi(y_t|y_{t-1}, s_t) + \eta Q(X, y_{1\cdots t}). \quad (37)$$

Specifically, the Q-function, in this paper, takes only the hidden state at time t and estimates the length of the remaining sequence. While decoding, they suggest an inference method that controls the length of the generated sequence as follows:

$$\hat{y}_t = \arg\max_y \log \pi(y|\hat{y}_{1\cdots t-1}, X) - \eta||(T-t) - Q_\Psi(s_t)||^2. \quad (38)$$

Recently, Li *et al.* [46] proposed an AC model that uses a binary classifier as the critic. In this specific model, the critic tries to distinguish between the generated summary and the human-written summary via a neural network binary classifier. Once they pre-trained the actor using CE loss, they start training the AC model alternatively using PG and the classifier score is considered as a surrogate for the value function. AC and PG were used also in the work of Liu *et al.* [136], where they combined AC and PG learning along with importance sampling to train a seq2seq model for image captioning. In this method, they used two different neural networks for Q-function estimation, i.e., Q_Ψ, and value estimation, i.e., $V_{\Psi'}$. They also used a mixed reward function that combines a weighted sum of ROUGE, BLEU, METEOR, and CIDEr measures to achieve higher performance on this task.

C. Current RL-Based Model Issues

Throughout this paper, we discussed various situations, where using RL provides a better solution than the traditional methods. However, utilizing RL methods creates its own training challenges, and in most of the cases, the improvement received from these models is not significant. In this section, we will discuss some of the issues that exist in current RL techniques used for seq2seq problems. As discussed in Section III, sample efficiency and high variance in RL models are among the main issues in applying them to seq2seq problems. Therefore, models such as RAML [175] and SPG [176] are proposed to provide a middle ground between the MLE and RL training. In RAML [175], a reward-aware perturbation is added to MLE, while in SPG [176], the reward distribution is utilized for the effective sampling of PG. Recently, Tan *et al.* [177] provided a general formulation that connects the MLE and RL training through entropy regularized policy optimization (ERPO). However, even these solutions suffer from their own problems. RAML arguably suffers from the exposure bias, while SPG requires a lot of engineering to work on a specific problem, and, as shown in Table III, that is why REINFORCE-based models, such as MIXER [28], are preferred in most of the current seq2seq problems.

Although REINFORCE-based models are simple to implement and provide better results, training these models is time-consuming, and the improvement over baselines is usually marginal. This is why in most of the current works, these models are only used for fine-tuning purposes.

Aside from these issues, there are problems inherent to specific applications that make it hard for researchers to combine RL techniques with current seq2seq models. For instance, in most of the NLP problems, the output or action space is massive compared to the size of actions in the robotic or game-playing problems. This is mostly due to the fact that in applications, such as machine translation, text summarization, and image captioning, the size of the output

is equal to the size of the vocabulary used during training. This is extremely high compared to the agents used in other applications, e.g., an agent that plays an Atari game requires deciding on usually less than 20 actions [178]. This will show the severity of this problem and the reward sparsity issue that exists in these applications.

Moreover, most of the current seq2seq models that use RL training rely on well-defined reward functions, such as BLEU or ROUGE, for providing feedback to the model. Although these are the standard metrics for evaluating various seq2seq models, relying on them creates a different set of problems. For instance, in abstractive text summarization, ROUGE and BLEU scores are being used as the standard metrics for evaluating the summarization models. However, a good abstractive summary will definitely have a low ROUGE and BLEU score. This problem could be further investigated and possibly improved by inverse RL (IRL) [179] by forcing the model to learn its own rewarding function. However, to the best of our knowledge, no work has been done in this area.

Recently, new methods are introduced for game playing using the RL algorithm, which combines the best performing models in this area and applies some of the best practices used in previous models to achieve the state-of-the-art results. Rainbow [180] and Quantile Nets [181] are among such frameworks. In Rainbow [180], the authors combine DDQN, prioritized experience buffer, dueling net, multi-step learning (using step-based reward rather than general reward), and distributional RL to achieve state of the art in 57 games in the Atari 2600 framework. A similar ensembling method could also be useful to be applied for seq2seq tasks, but this is also left for future work.

V. RLSeq2Seq: An Open-Source Library for Training seq2seq Models With RL Methods

As a part of this comprehensive study, we developed an open-source library that implements various RL techniques for the problem of abstractive text summarization. This library is made available at *www.github.com/yaserkl/RLSeq2Seq/*. Since experimenting each specific configuration of these models even requires few days of training on GPUs, we encourage researchers, who use this library to build and enhance their own models, to also share their trained model at this website. In this section, we explain some of the important features of our library. As mentioned earlier, this library provides modules for abstractive text summarization. The core of our library is based on a model called pointer-generator[38] [12] that itself is based on Google TextSum model.[39] We also provide a similar imitation learning used in training REINFORCE algorithm to train the function approximator. This way, we propose training our DQN (DDQN, Dueling Net) using a scheduled sampling, in which we start training the model in the beginning based on the ground-truth Q-values while we move on with the training process, and we completely rely on the function estimator to train the network. This could be seen as a pre-training step for the function approximator. Therefore, the model is guaranteed to start by using better ground-truth data since it is exposed to the true ground-truth values versus the random estimation it receives from the model itself. In summary, our library implements the following features:

1) adding temporal attention and intra-decoder attention that was proposed in [13];
2) adding scheduled sampling along with its differentiable relaxation proposed in [31] and E2EBackProb [28] for solving *exposure bias* problem;
3) adding adaptive training of REINFORCE algorithm by minimizing the mixed objective loss in (34);
4) providing SC training by adding the greedy reward as the baseline;
5) providing AC training options for training the model using asynchronous training of Value Network, DQN, DDQN, and Dueling Net;
6) providing options for scheduled sampling for the training of the Q-Function in DQN, DDQN, and Dueling Net.

A. Experiments

To test the power of some of the studied models in this paper, we performed a range of various experiments using our open-source library. As mentioned in Section IV, most of the RL-based models play as a fine-tuning technique in seq2seq applications. Thus, we first pre-train our model for 15 epochs using only CE loss and then add the RL training for another ten epochs. Our experiments follow the same setup like the one described in the pointer-generator paper [12], and we only show the results after activating the coverage mechanism. We activate the coverage mechanism only for the last epoch and select the best model using the evaluation data. We use a linear scheduling probability as $\epsilon = step/total\ steps$, and we also use $\epsilon = 1$ after activating the coverage so that the model completely relies on its own output for the rest of training, and for E2EBackPropagation model, K is set to 4. All experiments are done using two NVIDIA P100 GPUs: one used for training the model and the other for selecting the best-trained model based on the evaluation data.

1) Analysis of the Results: Table V shows the results of our experiments based on the ROUGE score on this data set. All our ROUGE scores have a 95% confidence interval of at most ± 0.25 as reported by the official ROUGE script. In Table V, PG stands for pointer generation and SS stands for scheduled sampling. As shown in Table V, both the scheduled sampling model and the E2E model are superior to the pointer generator. We have also used our framework to train the self-critic PG (SCPG)-based model proposed by Paulus *et al.* [13]. However, as shown in Table V, although the SCPG improves the performance of the pointer-generator model, this improvement is very marginal. This result is totally in contrast with the result in the original paper [13] and shows that SCPG, as claimed by the authors, will not greatly improve the performance of the pointer-generator model. One of the main reasons for this difference in the result of our experiment with the one in Paulus *et al.* [13] is that they use a completely different set of hyperparameters for training their model. For instance, the input for their encoder is 800 words, while in

[38] https://github.com/abisee/pointer-generator

[39] https://github.com/tensorflow/models/tree/master/research/textsum

TABLE V

ANALYSIS OF ROUGE F1-SCORE AFTER ACTIVATING COVERAGE ON TEXT SUMMARIZATION PROBLEM ON THE CNN/DM TEST DATA SET

Method	ROUGE			Training Time
	1	2	L	
PG	38.21	16.46	34.79	3-4 days
PG w. E2E	38.24	16.48	34.97	3-4 days
PG w. SS Argmax-Sampling	38.29	16.51	35.02	3-4 days
PG w. SS Argmax-Greedy	38.65	16.77	35.37	3-4 days
PG w. SCPG	38.77	16.98	35.32	6-7 days
Actor-Critic (Q-Learning) [54]	40.88	17.80	**38.54**	3-4 hours
DCA (SCPG) [174]	**41.69**	**19.47**	37.92	5-6 days

our default setting, for all our experiments, it is set to 400. Also, the vocabulary size is set to 150k and 50k for input and output, while our default is set to 50k for both input and output. Moreover, the size of hidden layers for encoder and decoder in their work is larger than our default values, and they also use a pre-trained GloVe word-embedding [182] for training their model. Finally, we are comparing all these PG-based models with an AC model proposed by Chen and Bansal [55], which holds the state-of-the-art result in text summarization in the CNN/DM data set. As shown in Table V, this model is superior to any of the PG-based models according to the ROUGE scores.

2) Analysis of the Training Time: In general, the pointer-generator framework requires more than three days of training for effective results, while this time will also be extended after adding the SCPG. On an average, each batch of training during MLE training will take 2–3 s, while once we add the SCPG loss, this time will be increased to 5–6 s, which means that the entire training time will be double after activation of the RL loss. On the other hand, the entire training time for the AC model before and after RL activation is only a few hours, which shows that not only it is superior in terms of the ROUGE score results but also the training process converges much faster than the other models.

VI. CONCLUSION

In this paper, we provided a general overview of a specific type of deep learning models called sequence-to-sequence (seq2seq) models and discussed some of the recent advances in combining training of these models with RL techniques. The seq2seq models are common in a wide range of applications from machine translation to speech recognition. However, traditional models in this field usually suffer from various problems during model training, such as inconsistency between the training objective and testing objective and *exposure bias*. Recently, with advances in deep RL, researchers offered various types of solutions to combine the RL training with seq2seq training for alleviating the problems and challenges of training seq2seq models. In this paper, we summarized some of the most important works that tried to combine these two different techniques and provided an open-source library for the problem of abstractive text summarization that shows how one could train a seq2seq model using different RL techniques.

REFERENCES

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NIPS*, 2014, pp. 3104–3112.

[2] S. Bengio *et al.*, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. NIPS*, 2015, pp. 1171–1179.

[3] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. EMNLP*, 2015, pp. 1412–1421.

[4] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*. [Online]. Available: https://arxiv.org/abs/1609.08144

[5] A. Vaswani *et al.*, "Tensor2tensor for neural machine translation," in *Proc. 13th Conf. Assoc. Mach. Transl. Amer.*, 2018, pp. 193–199.

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: https://arxiv.org/abs/1409.0473

[7] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. EMNLP*, 2014, pp. 1–10.

[8] S. Shen *et al.*, "Minimum risk training for neural machine translation," in *Proc. ACL*, vol. 1, Aug. 2016, pp. 1683–1692.

[9] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proc. EMNLP*, 2015, pp. 379–389.

[10] S. Chopra *et al.*, "Abstractive sentence summarization with attentive recurrent neural networks," in *Proc. NAACL-HLT*, Jun. 2016, pp. 93–98.

[11] R. Nallapati *et al.*, "Abstractive text summarization using sequence-to-sequence RNNS and beyond," in *Proc. SIGNLL*, 2016, pp. 280–290.

[12] A. See, P. J. Liu, and C. D. Manning, "Get to the Point: Summarization with pointer-generator networks," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2017, pp. 1073–1083.

[13] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," in *Proc. ICLR*, 2018, pp. 1–9.

[14] R. Nallapati, F. Zhai, and B. Zhou, "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents," in *Proc. AAAI*, Feb. 2017, pp. 3075–3081.

[15] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. ICASSP*, May 2013, pp. 6645–6649.

[16] D. Bahdanau *et al.*, "End-to-end attention-based large vocabulary speech recognition," in *Proc. ICASSP*, Mar. 2016, pp. 4945–4949.

[17] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proc. ICML*, Jun. 2016, pp. 173–182.

[18] S. Ö. Arik *et al.*, "Deep voice: Real-time neural text-to-speech," in *Proc. ICML*, Aug. 2017, pp. 195–204.

[19] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," *Comput. Sci.*, vol. 2015, pp. 2048–2057, Feb. 2015.

[20] O. Vinyals *et al.*, "Show and tell: A neural image caption generator," in *Proc. CVPR*, Jun. 2015, pp. 3156–3164.

[21] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. CVPR*, Apr. 2015, pp. 3128–3137.

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

[23] J. L. Elman, "Finding structure in time," *Cognit. Sci.*, vol. 14, no. 2, pp. 179–211, Mar. 1990.

[24] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Proc. Workshop Text Summarization Branchys Out*, 2004, pp. 74–81.

[25] K. Papineni *et al.*, "BLEU: A method for automatic evaluation of machine translation," in *Proc. ACL*, Jul. 2002, pp. 311–318.

[26] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proc. ACL Workshop Intrinsic Extrinsic Eval. Measures Mach. Transl. Summarization*, Jun. 2005, pp. 65–72.

[27] R. Vedantam, C. L. Zitnick, and D. Parikh, "CIDEr: Consensus-based image description evaluation," in *Proc. CVPR*, Jun. 2015, pp. 4566–4575.

[28] M. Ranzato *et al.*, "Sequence level training with recurrent neural networks," 2015, *arXiv:1511.06732*. [Online]. Available: https://arxiv.org/abs/1511.06732

[29] J. Su *et al.*, "Incorporating discriminator in sentence generation: A gibbs sampling method," in *Proc. AAAI*, Apr. 2018, pp. 1–27.

[30] F. Huszár, "How (not) to train your generative model: Scheduled sampling, likelihood, adversary?" 2015, *arXiv:1511.05101*. [Online]. Available: https://arxiv.org/abs/1511.05101

[31] K. Goyal, C. Dyer, and T. Berg-Kirkpatrick, "Differentiable scheduled sampling for credit assignment," in *Proc. ACL*, vol. 2, Jul. 2017, pp. 366–371.

[32] L. Yu *et al.*, "SeqGan: Sequence generative adversarial nets with policy gradient," in *Proc. AAAI*, vol. 31, Feb. 2017, pp. 2852–2858.

[33] K. Lin *et al.*, "Adversarial ranking for language generation," in *Proc. NIPS*, 2017, pp. 3155–3165.

[34] J. Guo *et al.*, "Long text generation via adversarial training with leaked information," in *Proc. AAAI*, 2018, pp. 1—27.

[35] T. Che *et al.*, "Maximum-likelihood augmented discrete generative adversarial networks," 2017, *arXiv:1702.07983*. [Online]. Available: https://arxiv.org/abs/1702.07983

[36] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. NIPS*, 2014, pp. 2672–2680.

[37] Y. Zhang *et al.*, "Adversarial feature matching for text generation," in *Proc. ICML*, Aug. 2017, pp. 4006–4015.

[38] Z. Shi *et al.*, "Toward diverse text generation with inverse reinforcement learning," in *Proc. IJCAI*, Jul. 2018, pp. 4361–4367.

[39] D. Bahdanau *et al.*, "An actor-critic algorithm for sequence prediction," in *Proc. ICLR*, 2017.

[40] S. J. Rennie *et al.*, "Self-critical sequence training for image captioning," in *Proc. CVPR*, Jul. 2017, pp. 7008–7024.

[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–2256, May 1992.

[42] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.

[43] S. Levine *et al.*, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2015.

[44] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J. Robot. Res.*, vol. 34, nos. 4–5, pp. 705–724, 2015.

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

[46] P. Li, L. Bing, and W. Lam, "Actor-critic based training framework for abstractive summarization," 2018, *arXiv:1803.11070*. [Online]. Available: https://arxiv.org/abs/1803.11070

[47] K. Arulkumaran *et al.*, "A brief survey of deep reinforcement learning," 2017, *arXiv:1708.05866*. [Online]. Available: https://arxiv.org/abs/1708.05866

[48] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*. [Online]. Available: https://arxiv.org/abs/1701.07274

[49] S. Team, "Deep learning for siri's voice: On-device deep mixture density networks for hybrid unit selection synthesis," *Apple Mach. Learn. J.*, vol. 1, p. 4, Apr. 2017.

[50] G. Klein *et al.*, "Opennmt: Open-source toolkit for neural machine translation," in *Proc. ACL*, 2017, pp. 67–72.

[51] M. X. Chen *et al.*, "The best of both worlds: Combining recent advances in neural machine translation," in *Proc. ACL*, 2018, pp. 76–86.

[52] T. Shi *et al.*, "Neural abstractive text summarization with sequence-to-sequence models," 2018, *arXiv:1812.02303*. [Online]. Available: https://arxiv.org/abs/1812.02303

[53] Q. Zhou *et al.*, "Selective encoding for abstractive sentence summarization," in *Proc. ACL*, Jan. 2017, pp. 1095–1104.

[54] J. Tan, X. Wan, and J. Xiao, "Abstractive document summarization with a graph-based attentional neural model," in *Proc. ACL*, Jan. 2017, pp. 1171–1181.

[55] Y.-C. Chen and M. Bansal, "Fast abstractive summarization with reinforce-selected sentence rewriting," in *Proc. ACL*, Jul. 2018, pp. 675–686.

[56] J. Lin *et al.*, "Global encoding for abstractive summarization," in *Proc. ACL*, Jul. 2018, pp. 163–169.

[57] W.-T. Hsu *et al.*, "A unified model for extractive and abstractive summarization using inconsistency loss," in *Proc. ACL*, Jul. 2018, pp. 132–141.

[58] Y. Xia *et al.*, "Deliberation networks: Sequence generation beyond one-pass decoding," in *Proc. NIPS*, 2017, pp. 1782–1792.

[59] I. V. Serban *et al.*, "Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus," in *Proc. ACL*, Jul. 2016, pp. 588–598.

[60] N. Mostafazadeh *et al.* "Generating natural questions about an image," in *Proc. ACL*, Aug. 2016, pp. 1802–1813.

[61] Z. Yang *et al.*, "Semi-supervised QA with generative domain-adaptive nets," in *Proc. ACL*, Sep. 2017, pp. 1040–1050.

[62] X. Yuan *et al.*, "Machine comprehension by text-to-text neural question generation," in *Proc. 2nd Workshop Represent. Learn. NLP*, Aug. 2017, pp. 15–25.

[63] S. Antol *et al.*, "VQA: Visual question answering," in *Proc. ICCV*, Dec. 2015, pp. 2425–2433.

[64] C. Xiong, V. Zhong, and R. Socher, "Dynamic coattention networks for question answering," in *Proc. ICLR*, 2016, pp. 116–120.

[65] Z. Yang *et al.*, "Stacked attention networks for image question answering," in *Proc. CVPR*, Jun. 2016, pp. 21–29.

[66] M. Zhu *et al.*, "A hierarchical attention retrieval model for healthcare question answering," in *Proc. World Wide Web Conf.*, May 2019, pp. 2472–2482

[67] O. Vinyals and Q. Le, "A neural conversational model," 2015, *arXiv:1506.05869*. [Online]. Available: https://arxiv.org/abs/1506.05869

[68] J. Li *et al.*, "A diversity-promoting objective function for neural conversation models," in *Proc. NAACL-HLT*, Jun. 2016, pp. 110–119.

[69] I. V. Serban *et al.*, "Building end-to-end dialogue systems using generative hierarchical neural network models," in *Proc. AAAI*, vol. 16, 2016, pp. 3776–3784.

[70] A. Bordes, Y.-L. Boureau, and J. Weston, "Learning end-to-end goal-oriented dialog," in *Proc. ICLR*, 2016, pp. 12–34.

[71] V. Zhong, C. Xiong, and R. Socher. (2018). *Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning*. [Online]. Available: https://openreview.net/forum?id=Syx6bz-Ab

[72] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," 2017, *arXiv:1711.04436*. [Online] Available: https://arxiv.org/abs/1711.04436

[73] R. Kiros, R. Salakhutdinov, and R. Zemel, "Multimodal neural language models," in *Proc. ICML*, Jan. 2014, pp. 595–603.

[74] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," 2014, *arXiv:1411.2539*. [Online]. Available: https://arxiv.org/abs/1411.2539

[75] X. Chen and C. L. Zitnick, "Mind's eye: A recurrent visual representation for image caption generation," in *Proc. CVPR*, Jun. 2015, pp. 2422–2431.

[76] J. Mao *et al.*, "Deep captioning with multimodal recurrent neural networks (m-RNN)," 2014, *arXiv:1412.6632*. [Online]. Available: https://arxiv.org/abs/1412.6632

[77] H. Fang *et al.*, "From captions to visual concepts and back," in *Proc. CVPR*, Jun. 2015, pp. 1473–1482.

[78] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. CVPR*, Apr. 2015, pp. 2625–2634.

[79] S. Venugopalan *et al.*, "Translating videos to natural language using deep recurrent neural networks," in *Proc. NAACL-HLT*, 2015, pp. 1494–1504.

[80] S. Venugopalan *et al.*, "Sequence to sequence—Video to text," in *Proc. ICCV*, Dec. 2015, pp. 4534–4542.

[81] S. Venugopalan *et al.*, "Improving LSTM-based video description with linguistic knowledge mined from text," in *Proc. EMNLP*, Nov. 2016, pp. 1961–1966.

[82] P. Sermanet *et al.*, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *Proc. ICLR*, 2014, pp. 31–40.

[83] D. Erhan *et al.*, "Scalable object detection using deep neural networks," in *Proc. CVPR*, Sep. 2014, pp. 2147–2154.

[84] R. Girshick, "Fast R-CNN," in *Proc. ICCV*, Dec. 2015, pp. 1440–1448.

[85] S. Ren *et al.*, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2015, pp. 91–99.

[86] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. ICML*, Jan. 2014, pp. 1764–1772.

[87] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand. (ASRU)*, Dec. 2015, pp. 167–174.

[88] H. Ze, A. Senior, and M. Schuster, "Statistical parametric speech synthesis using deep neural networks," in *Proc. ICASSP*, May 2013, pp. 7962–7966.

[89] Y. Fan *et al.*, "TTS synthesis with bidirectional LSTM based recurrent neural networks," in *Proc. ISCA*, 2014, pp. 54–69.

[90] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," 2014, *arXiv:1412.7755*. [Online]. Available: https://arxiv.org/abs/1412.7755

[91] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. NIPS*, 2015, pp. 2692–2700.

[92] A. Vaswani *et al.*, "Attention is all you need," in *Proc. NIPS*, 2017, pp. 6000–6010.

[93] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment," 2017, *arXiv:1704.01444*. [Online]. Available: https://arxiv.org/abs/1704.01444

[94] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proc. COLING*, 2014, pp. 69–78.

[95] R. Socher *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proc. EMNLP*, Aug. 2013, pp. 1631–1642.

[96] A. Conneau *et al.*, "Supervised learning of universal sentence representations from natural language inference data," in *Proc. EMNLP*, Sep. 2017, pp. 670–680.

[97] S. Kim *et al.*, "Semantic sentence matching with densely-connected recurrent and co-attentive information," 2018, *arXiv:1805.11360*. [Online]. Available: https://arxiv.org/abs/1805.11360

[98] B. Pan *et al.*, "Discourse marker augmented network with reinforcement learning for natural language inference," in *Proc. ACL*, Jul. 2018, pp. 989–999.

[99] N. FitzGerald *et al.*, "Semantic role labeling with neural network factors," in *Proc. EMNLP*, Sep. 2015, pp. 960–970.

[100] L. He, M. Lewis, and L. Zettlemoyer, "Question-answer driven semantic role labeling: Using natural language to annotate natural language," in *Proc. EMNLP*, Jul. 2015, pp. 643–653.

[101] D. Marcheggiani, A. Frolov, and I. Titov, "A simple and accurate syntax-agnostic neural model for dependency-based semantic role labeling," in *Proc. CONLL*, Aug. 2017, pp. 411–420.

[102] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proc. EMNLP*, Sep. 2017, pp. 1506–1515.

[103] M. Mintz *et al.*, "Distant supervision for relation extraction without labeled data," in *Proc. ACL*, Aug. 2009, pp. 1003–1011.

[104] X. Huang *et al.*, "Attention-based convolutional neural network for semantic relation extraction," in *Proc. ICLR*, Sep. 2016, pp. 2526–2536.

[105] P. Qin, W. Xu, and W. Y. Wang, "Robust distant supervision relation extraction via deep reinforcement learning," in *Proc. ACL*, Jul. 2018, pp. 2137–2147.

[106] C. Chen and V. Ng, "Chinese zero pronoun resolution with deep neural networks," in *Proc. ACL*, vol. 1, Aug. 2016, pp. 778–788.

[107] Q. Yin *et al.*, "Chinese zero pronoun resolution with deep memory network," in *Proc. EMNLP*, Sep. 2017, pp. 1309–1318.

[108] T. H. Trinh and Q. V. Le, "A simple method for commonsense reasoning," 2018, *arXiv:1806.02847*. [Online]. Available: https://arxiv.org/abs/1806.02847

[109] Q. Yin *et al.*, "Deep reinforcement learning for Chinese zero pronoun resolution," in *Proc. ACL*, Nov. 2018, pp. 569–578.

[110] P. Anderson *et al.*, "SPICE: Semantic propositional image caption evaluation," in *Proc. ECCV*, 2016, pp. 382–398.

[111] A. Axelrod, X. He, and J. Gao, "Domain adaptation via pseudo in-domain data selection," in *Proc. EMNLP*, Jul. 2011, pp. 355–362.

[112] K. M. Hermann *et al.*, "Teaching machines to read and comprehend," in *Proc. NIPS*, 2015, pp. 1693–1701.

[113] M. Grusky, M. Naaman, and Y. Artzi, "Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies," in *Proc. NAACL-HLT*, Jun. 2018, pp. 708–719. [Online]. Available: https://summari.es/newsroom.pdf

[114] D. Graff *et al.*, *English Gigaword*. Philadelphia, PA, USA: Linguistic Data Consortium, 2003, p. 1.

[115] P. Rajpurkar *et al.*, "Squad: 100,000+ questions for machine comprehension of text," in *Proc. EMNLP*, Nov. 2016, pp. 2383–2392.

[116] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," in *Proc. ACL*, Aug. 2018, pp. 784–789.

[117] M. Joshi *et al.*, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proc. ACL*, vol. 1, Jul. 2017, pp. 1601–1611.

[118] J. Tiedemann. (2018). *News from OpusžA Collection of Multilingual Parallel Corpora With Tools and Interfaces 1 Index of Subjects and Terms 13*. [Online]. Available: http://opus.nlpl.eu/

[119] J. Dodge *et al.*, "Evaluating prerequisite qualities for learning end-to-end dialog systems," 2015, *arXiv:1511.06931*. [Online]. Available: https://arxiv.org/abs/1511.06931

[120] C. Danescu-Niculescu-Mizil and L. Lee, "Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs," in *Proc. 2nd Workshop Cognit. Model. Comput. Linguistics*, Jun. 2011, pp. 76–87.

[121] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," in *Proc. ACL*, Jul. 2015, pp. 1470–1480.

[122] Y. Wang, J. Berant, and P. Liang, "Building a semantic parser overnight," in *Proc. ACL*, vol. 1, Jul. 2015, pp. 1332–1342.

[123] J. McAuley, R. Pandey, and J. Leskovec, "Inferring networks of substitutable and complementary products," in *Proc. KDD*, Aug. 2015, pp. 785–794.

[124] S. R. Bowman *et al.*, "A large annotated corpus for learning natural language inference," in *Proc. EMNLP*, Jul. 2015, pp. 148–158.

[125] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, 2018, pp. 1112–1122.

[126] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Comput. linguistics*, vol. 31, no. 1, pp. 71–106, Mar. 2005.

[127] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," in *Proc. ACL*, Jul. 2005, pp. 363–370.

[128] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[129] V. Ordonez, G. Kulkarni, and T. L. Berg, "Im2text: Describing images using 1 million captioned photographs," in *Proc. NIPS*, 2011, pp. 1143–1151.

[130] J. Xu *et al.*, "MSR-VTT: A large video description dataset for bridging video and language," in *Proc. CVPR*, Jun. 2016, pp. 5288–5296.

[131] D. L. Chen and W. B. Dolan, "Collecting highly parallel data for paraphrase evaluation," in *Proc. ACL*, Jun. 2011, pp. 190–200.

[132] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[133] V. Panayotov *et al.*, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. ICASSP*, Apr. 2015, pp. 5206–5210.

[134] W. Zaremba and I. Sutskever, "Reinforcement learning neural Turing machines–revised," 2015, *arXiv:1505.00521*. [Online]. Available: https://arxiv.org/abs/1505.00521

[135] T. Jie and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Proc. NIPS*, 2010, pp. 1000–1008.

[136] S. Liu *et al.*, "Improved image captioning via policy gradient optimization of spider," in *Proc. ICCV*, vol. 3, Mar. 2017, pp. 873–881.

[137] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, Jun. 2016, pp. 1928–1937.

[138] R. Munos *et al.*, "Safe and efficient off-policy reinforcement learning," in *Proc. NIPS*, 2016, pp. 1054–1062.

[139] M. Jaderberg *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," in *Proc. ICLR*, 2017, pp. 10–20.

[140] A. Gruslys *et al.*, "The reactor: A sample-efficient actor-critic architecture," in *Proc. ICLR*, Aug. 2018, pp. 1–17.

[141] Z. Wang *et al.*, "Sample efficient actor-critic with experience replay," in *Proc. ICLR*, 2017, pp. 11–25.

[142] R. S. Sutton *et al.*, "Policy gradient methods for reinforcement learning with function approximation," in *P NIPS*, 2000, pp. 1057–1063.

[143] G. Tucker *et al.*, "The mirage of action-dependent baselines in reinforcement learning," in *Proc. ICLR Workshop*, 2018.

[144] J. Schulman *et al.*, "High-dimensional continuous control using generalized advantage estimation," in *Proc. ICLR*, Jul. 2016, pp. 125–146.

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

[146] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[147] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Mach. Learn.*, vol. 16, no. 16, pp. 185–202, Sep. 1994.

[148] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. learn.*, vol. 8, nos. 3–4, pp. 293–321, 1992.

[149] T. Schaul *et al.*, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: https://arxiv.org/abs/1511.05952

[150] H. V. Hasselt, "Double q-learning," in *Proc. NIPS*, 2010, pp. 2613–2621.

[151] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Connectionist Models Summer School Hillsdale*. New York, NY, USA: Lawrence Erlbaum, 1993.

[152] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, Feb. 2016, pp. 15–30.

[153] Z. Wang *et al.*, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 2016, pp. 1995–2003.

[154] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Mach. Learn.*, vol. 75, no. 3, pp. 297–325, Jun. 2009.

[155] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, Jun. 2011, pp. 627–635.

[156] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," in *Proc. AAAI*, Jan. 2015, pp. 3024–3030.

[157] V. Mnih *et al.*, "Recurrent models of visual attention," in *Proc. NIPS*, 2014, pp. 2204–2212.

[158] L. Wang *et al.*, "A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization," in *Proc. IJCAI*, 2018, pp. 258–268.

[159] S. Narayan, S. B. Cohen, and M. Lapata, "Ranking sentences for extractive summarization with reinforcement learning," in *Proc. NAACL-HLT*, vol. 1, Jun. 2018, pp. 1747–1759.

[160] Z. Li *et al.*, "Paraphrase generation with deep reinforcement learning," in *Proc. EMNLP*, 2018, pp. 3865–3878.

[161] R. Pasunuru and M. Bansal, "Reinforced video captioning with entailment rewards," in *Proc. EMNLP*, Jul. 2017, pp. 979–985.

[162] Y. Zhou, C. Xiong, and R. Socher, "Improving end-to-end speech recognition with policy learning," in *Proc. ICASSP*, Apr. 2018, pp. 5819–5823.

[163] J. Li *et al.*, "Deep reinforcement learning for dialogue generation," in *Proc. EMNLP*, 2016, pp. 1192–1202.

[164] M. Hu, Y. Peng, and X. Qiu, "Reinforced mnemonic reader for machine comprehension," 2017, *arXiv:1705.02798*. [Online]. Available: https://arxiv.org/abs/1705.02798

[165] D. He *et al.*, "Decoding with value networks for neural machine translation," in *Proc. NIPS*, 2017, pp. 177–186.

[166] L. Zhang *et al.*, "Actor-critic sequence training for image captioning," 2017, *arXiv:1706.09601*. [Online]. Available: https://arxiv.org/abs/1706.09601

[167] 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.

[168] J. Li, W. Monroe, and D. Jurafsky, "Learning to decode for future success," 2017, *arXiv:1701.06549*. [Online]. Available: https://arxiv.org/abs/1701.06549

[169] Y. Wu and B. Hu, "Learning to extract coherent summary via deep reinforcement learning," in *Proc. AAAI*, 2018, pp. 11–20.

[170] J. Kreutzer, J. Uyheng, and S. Riezler, "Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning," in *Proc. ACL*, vol. 1, Jul. 2018, pp. 1777–1788.

[171] C. Liang *et al.*, "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision," in *Proc. ACL*, Jul. 2017, pp. 23–33.

[172] S. Yeung *et al.*, "End-to-end learning of action detection from frame glimpses in videos," in *Proc. CVPR*, Jun. 2016, pp. 2678–2687.

[173] Y. Keneshloo, N. Ramakrishnan, and C. K. Reddy, *Deep Transfer Reinforcement Learning for Text Summarization*. Philadelphia, PA, USA: SIAM, 2019, pp. 675–683.

[174] I. Higgins *et al.*, "Darla: Improving zero-shot transfer in reinforcement learning," in *Int. Conf. Mach. Learn.*, Jul. 2017, pp. 1480–1490.

[175] M. Norouzi *et al.*, "Reward augmented maximum likelihood for neural structured prediction," in *Proc. NIPS*, 2016, pp. 1723–1731.

[176] N. Ding and R. Soricut, "Cold-start reinforcement learning with softmax policy gradient," in *Proc. NIPS*, 2017, pp. 2817–2826.

[177] B. Tan *et al.*, "Connecting the dots between MLE and RL for sequence generation," 2018, *arXiv:1811.09740*. [Online]. Available: https://arxiv.org/abs/1811.09740

[178] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/abs/1312.5602

[179] B. D. Ziebart *et al.*, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, 2008, pp. 1433–1438.

[180] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI*, Aug. 2018, pp. 1458–1469.

[181] W. Dabney *et al.*, "Implicit quantile networks for distributional reinforcement learning," 2018, *arXiv:1806.06923*. [Online]. Available: https://arxiv.org/abs/1806.06923

[182] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. EMNLP*, Jul. 2014, pp. 1532–1543.

[183] A. Celikyilmaz *et al.*, "Deep communicating agents for abstractive summarization," in *Proc. ACM*, Sep. 2018, pp. 1662–1675.

Yaser Keneshloo received the Ph.D. degree from the Department of Computer Science, Virginia Tech, Arlington, VA, USA, in 2019.

He is currently a Senior Manager of the Advance Data Science Team, Marriott International, Bethesda, MD, USA. His current research interests include deep reinforcement learning, text summarization, and predictive modeling.

Tian Shi received the Ph.D. degree in physical chemistry from Wayne State University, Detroit, MI, USA, in 2016. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Virginia Tech, Arlington, VA, USA.

His current research interests include data mining, deep learning, topic modeling, and text summarization.

Naren Ramakrishnan received the Ph.D. degree in computer sciences from Purdue University, West Lafayette, IN, USA.

He is currently the Thomas L. Phillips Professor of Engineering with Virginia Tech, Arlington, VA, USA. He directs the Discovery Analytics Center, a university-wide effort that brings together researchers from computer science, statistics, mathematics, and electrical and computer engineering to tackle knowledge discovery problems in important areas of national interest. His work has been featured in The Wall Street Journal, *Newsweek*, *Smithsonian Magazine*, PBS/NoVA Next, The Chronicle of Higher Education, and *Popular Science*, among other venues.

Dr. Ramakrishnan serves on the editorial boards of the IEEE COMPUTER, *ACM Transactions on Knowledge Discovery from Data*, *Data Mining and Knowledge Discovery*, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and other journals.

Chandan K. Reddy (S'02–M'07–SM'14) received the M.S. degree from Michigan State University, East Lansing, MI, USA, and the Ph.D. degree from Cornell University, Ithaca, NY, USA.

He is currently an Associate Professor with the Department of Computer Science, Virginia Tech, Arlington, VA, USA. He has published over 110 peer-reviewed articles in leading conferences and journals. His current research interests include data mining and machine learning with applications to healthcare analytics and social network analysis. His research is funded by the National Science Foundation, the National Institutes of Health, the Department of Transportation, and the Susan G. Komen for the Cure Foundation.

Dr. Reddy is a Life Member of the Association for Computing Machinery (ACM). He received several awards for his research work, including the Best Application Paper Award at the ACM Special Interest Group on Knowledge Discovery and Data Mining Conference in 2010, the Best Poster Award at the IEEE Conference on Visual Analytics Science and Technology in 2014, the Best Student Paper Award at the IEEE International Conference on Data Mining in 2016, and was a finalist of the INFORMS Franz Edelman Award Competition in 2011. He was the PC Co-Chair of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining in 2018. He is also an Associate Editor of the *ACM Transactions on Knowledge Discovery and Data Mining*.