Deep Q-Learning for Text Generation

Stanford CS224N Custom Project

Felix Meng
ICME
Stanford University
felixmg@stanford.edu

Peranut Nimitsurachat
ICME
Stanford University
phee27@stanford.edu

Liwen Ouyang
ICME
Stanford University
ouyang42@stanford.edu

Abstract

When training a decoder, teacher forcing method will often lead to exposure bias when generating new text. By training with the teacher forcing method, the model will only minimize a maximum-likelihood loss at each individual decoding step given the golden context from real human generated text. In this work, we use Reinforcement Learning (deep Q network) to fine tune the decoder. Instead of selecting the next token with highest probability, we train reinforcement learning decoder to select from three actions–adding word, removing word, and replacing words. The proposed approach incorporates a classifier to provide a state vector encapsulating information from both the input sentence and the currently generated sentence, allowing for more controlled text generation. Additionally, the concept of backtracking for sentence generation is introduced through delete word and replace word, enabling agents to backtrack and generate words. This approach is a departure from traditional decoding methods, and we argue that our framework can be applied to other text generation tasks with minor modifications. Results show that the proposed approach generates more coherent and grammatically accurate sentences, although it may have slightly lower performance metrics than traditional approaches.

1 Introduction

We discovered that the approach described in the previous papers, which involved treating every word in the vocabulary as a possible action for the reinforcement learning problem, was not feasible. The sheer number of possible actions and the sparsity of rewards along the action space made it nearly impossible for the model to converge. Even if only the top 40 words were considered for a reasonable reward, the probability of the model choosing one of these actions would be extremely low. The probability of choosing the 40 actions will just be $\frac{40}{|V|} \approx 0.0007$, making it impossible for the agent to learn a meaningful policy during training. In addition, it'll be hard to reward the agent even if they have chosen the correct action. As a result, we revised our approach and developed a new framework for the Reinforcement Learning Decoder that limits the action space. We also focused specifically on text summarization, rather than other text generation tasks due to time constraints. However, we argue that our approach and architecture can be applied to other text generation tasks by just making small modifications in our constraint. Our method goes beyond the traditional approach of simply considering the previous sentence and calculating its probability. Instead, we incorporate a classifier that provides a state vector encapsulating information from both the input sentence and the currently generated sentence. This innovative approach allows for more controlled text generation by enabling the definition of specific criteria such as sentiment and topic, which the agent can use to guide its decisions. Furthermore, our method introduces the concept of backtracking for sentence generation. This means that our agents can now regret their previous decisions and backtrack to generate new, more appropriate sentences. This is a departure from traditional decoding methods like beam search, top-k, or top-p sampling, which do not allow for such flexibility. By incorporating backtracking, we can ensure that our agents are constantly exploring different possibilities and making the most optimal decisions based on the current state of the environment.

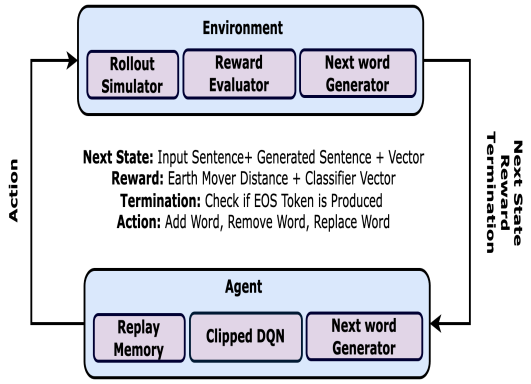

Figure 1: Overall architecture of modified transformer model

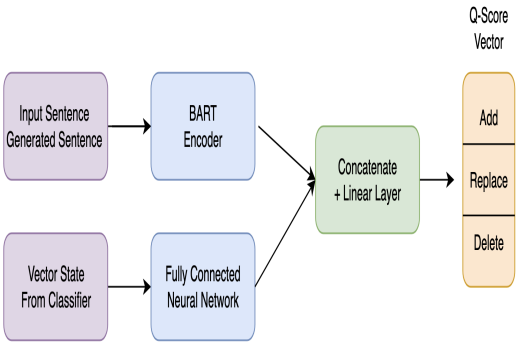

Figure 2: Deep Q Learning Framework

2 Related Work

We are unaware of any prior works that has used Deep Q Neural Network as a decoder like our proposed method. However, there are some literature that explore similar problems. For example, QUARK model[1] uses reward and penalty functions to fine-tune the model on what not to do. This technique enables the model to remove toxic language or offensive texts from its generated output while staying close to the original language model. Quark model is similar to our proposed method since it uses reward function to fine-tune the model. Our proposed method, however, is more complicated since, in addition to rewards and penalties, our model also uses Deep Q Neural Network to select the best next token through three variable actions (add word, replace word, and remove word).

Furthermore, reinforcement learning has also been used for text generation[2]; this cited work proposes reward-shaping strategies to address the challenges of sparse rewards and large action spaces. This method will provide dense rewards to each generated toke and allow the model to converge easier. Narrowing the number of actions and making the rewards more compact have been the approach that we use to improve our model throughout the project.

In order to limit overestimation of Deep Q Networks and overcome poor policy estimation, we also adopt the technique that utilizes two Deep Q Networks and take the one with lower value[3]. Inspired by this approach, we created two architectures of Deep Q Neural Networks to learn the optimal policy from given states.

3 Approach

3.1 Reinforcement Learning Problem Formation

To provide a comprehensive understanding of the reinforcement learning problem, we will first present an overview, and then divide the subsequent sections into environment and agent to provide a more detailed description.

- **Environment**- Our environment comprises four neural networks that are fine-tuned for specific tasks: fluency classification, sentiment classification, grammatical classification, and topic classification. Additionally, we have fine-tuned a BART Seq2Seq model for text summarization with the gigaword dataset, although we can swap it out with other models as needed for different tasks. We'll use the vector states from the classifiers to evaluate reward and generate next state. The BART Seq2Seq model will provide the top 50 candidate words when the action add word and remove word are called.
- **State**- The state will be the input sentence, generated sentence and the vector states from the reward evaluator.

- **Action**- The actions space will simply be 3, which includes add words, remove words and replace words. We'll query the fine-tuned model to generate text and sample its top 50 candidates word. Then we'll use the chosen word as the word to add.
- **Reward**- The reward will be divided into two components. If a complete sentence has not yet been generated, we will simulate a sentence using the simulator. We will then pass the simulated sentence and input sentence through our classifiers model, which will output corresponding scores. The reward will be calculated as the dot product between the cross entropy score and a scalar vector (The scalar vector can be tuned by the user depending on the task that focus on). Once a complete sentence is generated, we will evaluate the cross entropy loss between the vector state of the generated sentence and the output sentence. We will also calculate the sentence mover distance between the generated sentence and the output sentence to determine the reward.
- **Policy** - Since there are infinite possible states for this problem, we will not be able to store all the Q(s,a) values. Therefore, we proposed to use Deep Q Neural Network which serves as a surrogate to approximate the Q(s,a) value. Given a state, the Deep Q Neural network will output $v \in R^3$ that corresponds to the 3 actions score.

3.2 Environment

3.2.1 Rewards From Classifiers

The reward is computed using the outputs of the classifiers. These classifiers will take in the generated sentence s_r from the RL agent and the input sentence s_i and output raw score vectors. These row score vectors will then be used to compute a reward for some given action state pair. For fluency F and Grammatical Classifiers G, the outputs will be two probabilities corresponding to positive and negative fluency or grammar of the generated sentence. We want to incur punishment for actions that lead to incoherent or grammatically incorrect sentences. For the sentiment S and topic T classifier, we want the outputs of these two classifiers to be similar on both s_i and s_r. I.e. the generated sentence should have similar sentiments and topic as the input sentence. Therefore we design our raw reward vector to be:

$$\mathbf{r} = \begin{bmatrix} -\log(F(s_r)_{negative}) \\ -\log(G(s_r)_{negative}) \\ \text{CrossEntropy}(T(s_r), T(s_i)) \\ \text{CrossEntropy}(S(s_r), S(s_i)) \end{bmatrix}$$

The final reward is defined as:

$$r = \alpha^T \mathbf{r} + m(s_t, s_r)$$

where α is a hyperparameter we can adjust to specify how much the agent should care about, say, the topic of the generated sentence should match the topic of the input sentence. m is some metric we can use to compare the generated sentence s_r and the ground truth s_t

3.2.2 Rollout Simulator

To address the limitation of the classifiers only taking in a complete sentence as input, we have implemented a rollout simulator inspired by reinforcement learning. This simulator utilizes the GPT2 model to auto-fill a sentence once a subset of the sequence is generated. By using this method, we can estimate the future trajectory of taking an action and evaluate the scores and rewards with respect to the simulated full sentence. This allows us to measure rewards even when adding tokens to an incomplete sentence. Choosing the top k simulations is a crucial aspect of implementing the Rollout Simulator. If we always choose the best k words from the GPT2 decoder, then the agent will never choose to add word as the trajectories are already optimal. On the other hand, if we choose a large value of k, such as 1000, the agent will only choose to add word because adding a word will always yield better reward then removing a word. Therefore, finding the optimal value of k is essential to achieving good performance. After several experiments, we determined that choosing the top 100 simulations strikes a good balance between add word and remove word, which allows the agent to learn meaningful policies.

3.3 Agent

3.3.1 Q Learning

To begin, we have instantiated a Q neural network that takes both a sentence and a vector state as inputs and outputs a 3-dimensional vector. Our Q network architecture consists of a Transformer with an attention mechanism and a fully connected neural network. To combine the two outputs, we concatenate them and input them into another linear layer. To save training time, we initialized the Transformer using the weights and architecture from the BART Encoder. This network will be trained using the Clipped Double Q Learning that we'll describe later. We've also defined the loss as Huber Loss, which is a combination of the L1 loss and L2 loss, meaning that it behaves like L1 loss when the error is small, and like L2 loss when the error is large to prevent the model from getting stuck in local minima.

3.3.2 Clipped Double Q Learning

Initially, we used traditional Q-learning as the baseline approach for our deep reinforcement learning framework. However, we observed that during the training process, the learning was unstable and the loss fluctuated to values that were close to 1000. This motivated us to explore alternative algorithms that could improve convergence and stability. We turned to Double Clipped Q-learning, which is a variant of the standard Q-learning algorithm that has shown promise in reinforcement learning tasks. With Double Clipped Q-learning, we instantiated two Deep Q Networks and took the minimum value between the pair of networks to limit overestimation.[4] This approach helped to reduce the variance of the Q-value estimates and improved the stability and speed of learning, s the Huber loss only fluctuated between values 0 to 30 most of the time. Attached is the psuedocode for the described approach. In addition, we also applied experience replay by defining a replay buffer that is a First in First Out Deque, we'll store the transition set of (states, actions, rewards, next states, terminations) into the Deque. During training, the agent samples a batch of experiences randomly from the replay buffer and uses them to update the Q-value function. By using a random sample of experiences, the agent can avoid learning from correlated experiences, which can lead to unstable and inefficient learning.[4] Experience replay also allows for more efficient use of experience by reusing it multiple times for learning.

Algorithm 1 Pseudocode For Updating Agent

Require: DQN1 $\leftarrow$ DQN(*Parameters,Learning Rate)
Require: DQN2 $\leftarrow$ DQN(*Parameters,Learning Rate)
States,Actions,Rewards,Next_States,Terminations= ReadFromReplayBuffer(batch_size)
Curr_Q1= DQN1(States)[Actions]
Curr_Q2= DQN2(States)[Actions]
Next_Q1=DQN1(Next_States)
Next_Q2=DQN2(Next_States)
Next_Q = Rewards + $\gamma \min_{i=1,2}$ Next_$Q_i \max_a(s', a)$
$Loss_1$ = HuberLoss(Curr_Q1,Next_Q)
$Loss_2$ = HuberLoss(Curr_Q2,Next_Q)
Perform Gradient Descent on Both Models

3.4 Training the Decoder Agent

Before discussing about the detailed training process, we'll first provide an overview of our pipeline.

1. Fine-tune an autoencoder-decoder model such as BART on a large text corpus, then freeze the model and use it to query for the top 50 candidates for each generated token.
2. Create a double Q agent with attention mechanism that takes in a state and can perform one of three actions: add a word from the top 50 candidates, remove a word, or replace a word.
3. Add experience replay to the deep Q network to stabalize learning process.

4. Use a transformer layer to process the current sequence and a linear layer to process a vector of different classifiers such as coherence, fluency, grammar, and sentiment. Concatenate the outputs and use them as input to the Q network, which has an output action space of 3.
5. Use top k from GPT2 as a rollout method for the chosen action and evaluate the scores compared to the target sentence.

To train our reinforcement learning (RL) agent, we will first initialize several key components, including the environment, replay memory, agent, classifier, and deep Q network. Our chosen batch size will be 128, with a maximum action limit of 50 for sentence generation and a maximum replay memory size of 1000. The learning rate for the deep Q network will be set at 5e-5. For each input sentence and output sentence pair, we will initiate a corresponding environment. To choose the next action, our agent will use a decaying epsilon-greedy exploration strategy, beginning with a random action selection with $\epsilon = 0.5$. The epsilon decay rate will be set to 0.01, gradually reducing the value of ϵ to a minimum of $\epsilon_{min} = 0.01$. To select an action, our model will take the average of the output from the two deep Q networks, and output the argmax of the average. This approach allows us to balance exploration and exploitation during the learning process, while enabling our agent to make informed decisions based on its previous experiences stored in replay memory.

After selecting the appropriate action, our agent will feed it into the environment, which will in turn produce a new state, reward, and termination status for the updated action. We will evaluate the termination condition by checking whether an end-of-sentence (eos) token is produced. The new state will consist of the action applied to the previously generated sentence, as well as the new vector state obtained from the classifier. This updated state will provide our agent with a more accurate representation of the current state of the environment. The reward will be calculated according to the method outlined in section 3.2.1, taking into account various factors such as the fluency, coherence, and relevance of the generated sentence to the desired output. By using this reward structure, we can incentivize our agent to generate sentences that are both grammatically correct and semantically meaningful, while also ensuring that they are relevant to the desired output.

To update our neural networks during the reinforcement learning process, we will first evaluate the Q value for both the current and next state batches using the two neural networks. We will then calculate the expected Q value, which will be defined as the batch of rewards plus the minimum of the maximum action of the next Q value from both models. Once we have obtained the expected Q value, we will evaluate the Huber loss between the two neural networks and backpropagate the loss to update their parameters. This approach allows us to learn from the previous experiences stored in replay memory and adjust our models accordingly. By minimizing the difference between the predicted and expected Q values, we can ensure that our agent is making optimal decisions based on the current state of the environment. Alg 2 is a Pseudocode of the training process.

Algorithm 2 Pseudocode For Training RL Agent

```
Require: Initialize Environment, ReplayMemory, Agent, Classifier, DeepQNetwork
Replay_Memory ← Replay_Memory(Capacity)
DQN ← DQN(*Parameters,Learning Rate)
Classifier ← Classifier( FineTuned_Classifier_Models)
Agent ← Agent(Fine_Tuned_Model,DQN,Replay_Memory)
batch_size ← 128
for (Input,Target) in Data do
    Environment ← Env(pretrained model,input,target,classifier)
    state ← Env.initial_state()
    for epoch in Max Action Length do
        action ← Agent.action(state)
        next_state,reward,termination ← Env.step(action)
        state ← next_state
        agent.replay_buffer.push(state,action,next_state,reward,termination)
        if len(agent.replay_buffer)> batch_size then
            agent.update(batch_size)
        end if
    end for
end for
```

4 Experiments

4.1 Data

We use Gigaword dataset throughout this project[5]. This dataset contains the document and corresponding summary collected from around 4 million articles. Both of the baseline and our modified transformer models are trained to generate the headline (summary) given the full document from the dataset. We replaced all the numbers with "#" and replaced all the unknown words with unknown tokenizer "UNK". An example of input and output pair looks like this:

- **Input**:scottish power plc said tuesday it agreed to be bought by spain 's iberdrola sa in a deal that values the british utility at ##.# billion pounds -lrb- UNK .# billion.
- **Output**: scottish power accepts takeover bid from spain's iberdrola

4.2 Evaluation method

When evaluating the model, we use two evaluation matrices– sentence mover distance and Rouge score (F1). These matrices measure the similarity of generated sentences from models (baseline and our model) and the actual labels. The Rouge score we use will measure the number of matching '1-grams' between our model-generated text and a true label. We'll compare our generated sentence with the greedily decoded baseline method, where the decoder always chooses the word with the highest probability.

4.3 Experimental details

We evaluate and compare the performance of the baseline and our model on a hold-out test set with 100 data points.

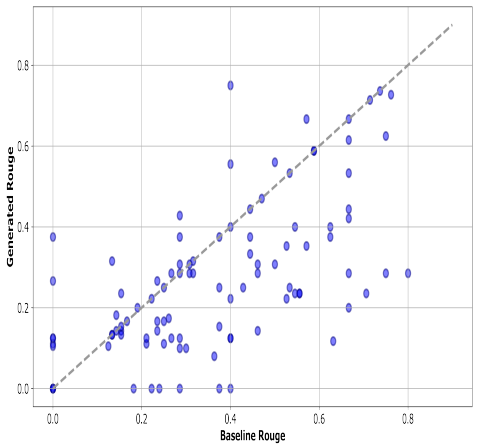

Figure 3: Scatter plot of Rouge scores of baseline and our model on the test set

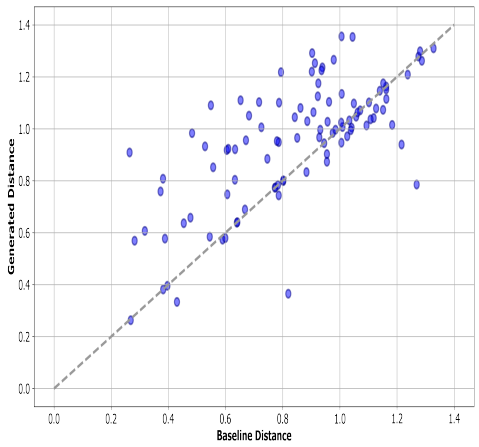

Figure 4: Scatter plot of mover distances of baseline and our model on the test set

Metric	Baseline	Our model
Rouge Scores	0.3655	0.2705
Mover Distance	0.8469	0.8844

Table 1: Summary of mean of Rogue scores and mover distances in test set

4.4 Results

After training our modified model on the training set, we monitor the Huber Backward Loss of the two models during the last 19,000 epochs in Fig. 6. The Huber Backward Loss is fluctuating a lot less compared to using the traditional Deep Q learning approach which fluctuates to 1000. Furthermore, we also evaluate the performance of our model on the test set and record the sentence mover distances of the generated sentences and baseline sentences to the actual labels. The scatter plot of these distances is shown in Fig. 4. We also calculate the difference in sentence mover distances between baseline and generated sentences to the actual labels and plot the distribution in Fig. 5. We also monitor the rogue scores from the baseline and our model and plot them in Fig. 3.

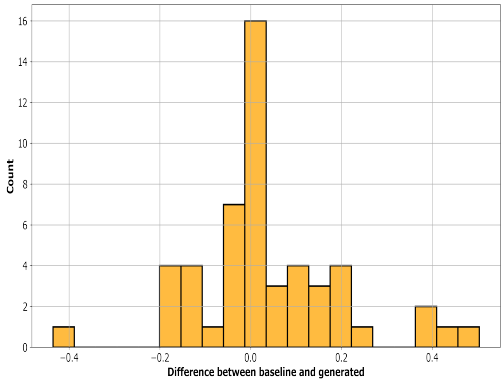

Figure 5: Distribution of the difference in mover distances between baseline and generated sentences to the actual labels

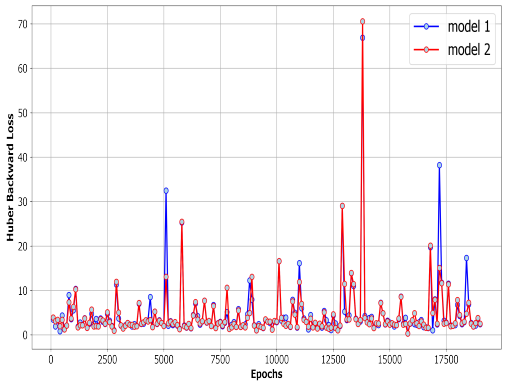

Figure 6: Huber Backward Loss from the last 19,000 epochs of training

The mover distances of baseline and generated sentences from Fig. 4 are what we expect. According to Fig. 4, most of the points (sentences in the test set) are scattering along the diagonal. Table. 1 also shows similar distances to true labels from baseline and our model. This shows that the generated sentences from our model are on par with the sentences from the baseline. Additionally, we also calculate the difference in mover distances between baseline and generated sentences to the actual labels. The distribution of these differences is shown in Fig.5. We can see that these differences are normally distributed and centered around zero. This again shows that our model is as effective as the baseline.

Finally, we also monitor the Rouge scores (F1) of the baseline and our model as shown in Fig. 3. This, however, does not align with our expectation since more than half of the scores are below the diagonal. This means that the baseline model slightly outperforms our model. We believe that this stems from the fact that our summaries usually use the new words outside of the document when generating summaries while the baseline usually sticks to with the words from the original document. Therefore, the number of overlapping words between generated summaries from baseline and our model are usually low; this, in turn, results in higher Rouge scores of the baseline model. Furthermore, due to limited computing resources, we can only train the model for a limited number of epochs. With more resources, we should be able to improve the Rouge score of our model.

5 Analysis

Our first attempt to this problem is training the RL decoder to select from the entire vocabulary. That is, we define the possible actions of our Deep Q Neural Networks to be the size of the entire vocabulary. This approach, however, fails; the baseline model can outperform our model by a large margin across all metrics. We therefore try to reduce the number of possible actions by allowing the model to only choose from three actions–adding, replacing and removing words. Furthermore, we also include information from other pre-trained classifiers, including grammatical rule and sentiment

of the original sentence, in the states of the decoder. Together, these modifications considerably improve the performance of our model. Our model can now generate summary sentences that make sense grammatically and capture the meaning of the original document. This improvement can also be seen through both of our evaluation matrices as described in the previous section.

	input sentence	baseline output	RL decoder output
1	scottish power plc said tuesday it agreed to be bought by spain 's iberdrola sa in a deal that values the british utility at ##.# billion pounds -lrb- UNK .# billion ; euro## .# billion -rrb- .	scottish power giant agrees to buy iberdrola sa	scottish power firm buys iberdrola sa in deal worth ##.# bln dlrs
2	hong kong's population will swell to #.## million in ## years, according to the latest projections by the city 's census and statistics department released thursday.	hong kong population to swell to #.## million in ## years	hong kong's population will swell to #.## million in ## years
3	a businessman in the pacific port city of vladivostok was blown to pieces monday when a bomb ripped through the elevator he was riding in what appeared to be an organized crime killing , a news report said.	bomb blast kills vladivostok man	automobile bomb kills vladivostok man

After analyzing the areas in which our model is outperformed by the baseline, we found that our approach often generates summaries using different words from the original document. This is because our model selects candidate words from the top 50 words, while the baseline model typically uses words directly from the document because it always chooses the word with the top probability. Consequently, the baseline model often achieves higher scores in our evaluation metric. However, upon reading the generated sequences, we discovered that our model tends to produce more coherent and grammatically accurate sentences from a human's perspective, despite scoring lower on the evaluation metric as shown in the above examples 1,2. This highlights that our reward is incentivizing the agent to generate sentences with better readability, grammatical accuracy, and coherency by allowing it to backtrack and sample from more words. However, this expressiveness of the RL decoder also leads to hallucination, i.e. generating seemingly correct sentences that are actually a wrong interpretation of the input, as shown in example 3. Future work can be done to mitigate the hallucination effect.

6 Conclusion

Our new approach has shown promising results in improving the performance of our model. By limiting the action space to only three options and incorporating pre-trained classifiers such as sentiment analysis and grammatical rules, we have seen a considerable improvement in the quality of the generated summaries. Our model is now capable of generating summary sentences that make sense grammatically and accurately capture the essence of the original document. These improvements are reflected in both our evaluation metrics (still almost on par with the baseline model) and user feedback. Despite these successes, we acknowledge that our model has limitations due to the limited amount of training data and computing resources available. With more resources and training time, we believe our model could potentially yield better result by careful choice of hyperparameter and exposure to more dataset. Additionally, we recognize that there are other external classifiers beyond grammatical and sentiment analysis that could potentially aid our model in generating high-quality summaries, and we believe that our model will have a more significant improvement if it is exposed to more information from a given sentence. In conclusion, our approach presents a promising direction for enhancing the performance of text generation models in diverse applications. Moving forward, we plan to further investigate this area of research by applying our model to more open-ended text generation tasks, such as creative writing. Additionally, we see many opportunities for exploring different reward evaluation techniques beyond the simple dot product between the vector state and a scalar vector that we are currently using. We are interested in studying the impact of varying the scalar vector on the output of the generated sentence, which could potentially lead to further improvements in our model. Ultimately, we believe that our approach has the potential to contribute to the development of more advanced and effective natural language generation systems.

References

[1] Ximing Lu, Sean Welleck, Jack Hessel, Liwei Jiang, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning, 2022.

[2] Bhargav Upadhyay, Akhilesh Sudhakar, and Arjun Maheswaran. Efficient reinforcement learning for unsupervised controlled text generation, 2022.

[3] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[4] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[5] David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword. *Linguistic Data Consortium, Philadelphia*, 4(1):34, 2003.