Reinforcement Learning for Language Models

Stanford CS224N Custom Project

Gary Qian
Stanford University
gqia189@stanford.edu

Wanqiao Xu
Stanford University
wanqiaox@stanford.edu

Paul Dupenloup
Stanford University
paul.dupenloup@stanford.edu

Abstract

Recent interest in Large Language Models (LLMs) and human alignment calls for effective finetuning methods. In this work, we investigate how reinforcement learning (RL) can be used to finetune the downstream performance of pretrained Language Models (LMs). Recently, on-policy RL algorithms have shown promise for text generation tasks. However, they face several empirical challenges, including (1) training instability due to the large action space and (2) sample inefficiency. In this paper, we explore methods to address both of these limitations. First, we implemented a variety of sampling techniques which effectively restrict the total action space without compromising performance, and which show significant improvement over vanilla proximal policy optimization (PPO). Second, we implemented an off-policy and value-based algorithm: Deep-Q Learning (DQN). We demonstrate the DQN can be applied to finetuning langauge models for downstream applications. However, futher exploration and tuning is required to determine whether it can achieve better sample efficiency compared to on-policy algorithms.

1 Key Information to include

- Mentor: Jesse Mu
- Sharing project: Paul is sharing this project with CS234
- Late days: Wanqiao and Gary have 4 late days left each (8 total) and Paul has 0. However we are submitting 3 days late. As per the Ed post on fractional late days, Wanqiao and Gary have enough days remaining, whereas Paul will incur a 1 day late penalty.

2 Introduction

With the recent surge of interest of LLMs, there has been much recent interest in finetuning pretrained models to align with human preference [1]. The finetuning process on natural language generation (NLG) tasks can be naturally framed as a sequential decision-making problem,where the model generates subsequent tokens based on context and aims to maximize sentence-level performance metrics. Indeed, there have been heated discussions around the effects and necessity of reinforcement learning with human feedback (RLHF) in fine-tuning LLMs.

The use of RL for NLG encounters several challenges, such as training instability due to the combinatorially large action space, high variance in automated language metrics, and reward hacking. Recently, the authors of RL4LM [2] proposed a novel on-policy RL algorithm named Natural Language Policy Optimization (NLPO). Notably, NLPO aims to address the large sample space issue by merging an on-policy PPO algorithm with a top-p truncation sampling technique. However, NLPO has two primary limitations that we aim to improve in this work.

First of all, naive truncation sampling methods like top-p can hinder high-quality generation. For example, top-p may truncate the action space too aggressively if only a few tokens take up the upper $p\%$ of the sampling distribution, resulting in repetitive text [3]. Meanwhile, if most tokens have low probabilities of getting sampled, top-p does not guarantee that the generated text will be of

high quality. Alternative sampling algorithms have been proposed that may produce more desirable behavior [3].

Second of all, the RL4LM library only implements on-policy algorithms, and does not implement any off-policy RL algorithms. Off-policy algorithms learn a policy function by evaluating or improving another policy, while on-policy algorithms learn a policy function from the same data that the policy is executed on. The former may offer several advantages, including better sample efficiency and more flexibility in data usage. These benefits could be especially critical for finetuning language tasks, since human labeling of data is prohibitively expensive or sometimes infeasible [4]. Inspired by this line of work, we aim to design RL algorithms that efficiently finetune pretrained LMs to align with preference metrics in NLP tasks. Specifically, we aim to explore methods to address both of the above-mentioned limitations. In this report, we detail our improvements to the sampling component of NLPO as well as our implementation of an off-policy algorithm.

3 Related Work

In the NLP context, RL has been used to improve model behavior in many tasks such as automatic translation [5, 6], question generation [7], summarization [8, 9], dialogue generation [10, 11], and text-based games [12, 13], to name a few. RL has recently been actively applied to ensure that LMs align with human preferences [8, 14, 15, 1, 16]. The use of RL in NLP is a fast-growing subfield with potential applications to various downstream tasks, yet it is still up for debate if RL is the right route or even necessary for improving language modeling and alignment.

On the RL side, large action space has been a challenge to efficient exploration. Common tasks in NLG involve generating the next token, which in principle has the whole vocabulary as candidates. When cast as an RL problem, the set of possible next tokens becomes a huge discrete action space. This poses challenges to many RL algorithms which are usually designed to deal with much smaller action spaces. Prior studies utilize sampling ideas [17, 18] and penalization of less likely actions in a critic [19, 20] to address the large action space problem. Since then, there have also been follow-up papers that attempt to eliminate actions [21] or generate small candidate action sets [22] for more efficient and stable exploration.

Many of these previous efforts at integrating RL and NLP exclusively focus on on-policy algorithms such as PPO. However, in certain settings, off-policy RL may be more sample efficient, as this approach allows for reusing datasets collected from prior interaction with a potentially different behavior policy [4]. Several prior papers have applied offline RL to NLP and more broadly to sequence generation problems [23] [24] [25], suggesting potential benefits of an off-policy approach.

4 Approach

Sampling algorithms. The first part of our project aims to improve the performance of the NLPO algorithm by implementing other truncation techniques on top of PPO. We describe them in detail below. In the following specifications, let $\mathcal{V}$ denote the vocabulary, $x^{(i)}$ the i-th token in an input sentence, $x_{<i}$ the prefix before the i-th token, θ the parameters of the model, h the entropy of the predictive distribution of the model, P the sampling probabilities, and $\mathcal{A}_{<i}$ the set of tokens allowed by a given sampling algorithm on the i-th step:

1. **Typical decoding** [26]: This approach extends the top-p method. The algorithm sorts tokens in the vocabulary in order of $|h_{\theta,x_{<i}} + \log p_\theta(x|x_{<i})|$. Given a fixed probability p, the algorithm then takes j highest-scoring tokens to cover p percent of the distribution. The allowed set of tokens on the i-th step is $\mathcal{A}_{x_{<i}} = \{x^{(1)}, ..., x^{(j)}\}$.
2. **ϵ-sampling** [3]: This algorithm truncates any token with probability no greater than ϵ. The allowed set of tokens on the i-th step is: $\mathcal{A}_{x_{<i}} = \{x \in \mathcal{V} : P_\theta(x|x_{<i}) > \epsilon\}$.
3. **η-sampling** [3]: This algorithm extends ϵ-sampling by truncating tokens below a minimum ϵ value and an entropy-dependent probability threshold. The allowed set of tokens can be expressed as follows (where α and ϵ are hyperparameters):

$$\mathcal{A}_{x_{<i}} = \{x \in \mathcal{V} : P_\theta(x|x_{<i}) > \eta, \eta = \min(\epsilon, \alpha \exp(-h_{\theta,x_{<i}}))\}.$$

4. **Mirostat** [27]: This algorithm uses an adaptive top-k sampling algorithm to actively tune the value of k, so as to maintain the perplexity of the generated text at a desired value. Specifically, the algorithm (1) estimates a value of s using minimum mean-squared errors (assuming tokens follow Zipf's law) and (2) estimates the value of k as a function of the estimated s and a target surprise value (μ). The allowed set of tokens on the i-th step is $\mathcal{A}_{x_{<i}} = \{x^{(1)}, ..., x^{(k)}\}$ where $k = ((s+1)2^{\mu})/(1-|\nu|^{-(s+1)}))^{\frac{1}{s}}$

The baselines we compare to are vanilla PPO and NLPO, where the latter uses a top-p sampling technique on top of PPO to truncate the action set. Given a fixed probability p, the top-p algorithm truncates tokens that are outside the mimimal set of (most probable) tokens that accounts for at least p percent of the distribution [3]. Note that tokens with up to $(1-p)$ probability may be truncated simply because other high-probability tokens cover probability p, which is an important limitation of this baseline sampling algorithm.

Delayed masking strategy. On a high-level, the NLPO algorithm masks out actions using top-p sampling. The implementation of NLPO in RL4LM contains two distinct masking strategies. Both train an unmasked policy. One strategy then masks out some actions under this policy at generation. The other strategy is to maintain a temporally static masked policy, and generate new tokens under the masked policy. The masked policy is synced with its unmasked counterpart every $n > 1$ training steps. In the following, we will refer to the first strategy with perfect syncing as usual, but call the second strategy with delayed syncing as "learned" versions of the sampling method, as it can be seen as *learning* a separate masked policy.

We have leveraged the existing codebase for the NLPO algorithm [28], as well as a stand-alone codebase Transformer Reinforcement Learning X repository (TRLX) [29]. RL4LM has NLPO with top-p and learned top-p implemented. TRLX has vanilla PPO implemented but does not have any existing implementation of sampling methods. Thus, we re-implemented top-p and learned top-p on TRLX. The implementation of all other sampling algorithms described above, as well as their learned counterparts, is entirely our own and coded from scratch in both codebases.

Off-Policy Algorithm. In the second part of our project, we implement Deep Q-Network (DQN) [30], a vanilla off-policy RL algorithm, catered to language generation. The goal of DQN is to train an action-value function $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ parametrized by a deep neural network.

Vanilla DQN performs ϵ-greedy exploration. At each time step, a random action is selected with probability ϵ, and a greedy action is selected with probability $1-\epsilon$. The greedy action is one that maximizes the action-value function given the current state s. In the NLP context, the action a is the next token chosen from the set of possible next tokens in our vocabulary $\mathcal{V}$, and the state s corresponds to the context embedding of tokens seen thus far. Then, a next state s' and a reward r are observed. The reward r corresponds to a textual metric (e.g. BLEU score) that depends on the particular NLG task. The loss function typically used is the temporal-difference (TD) loss, which computes the squared difference between the value of a given state-action pair and the value of the bootstrapped best future action (known as the target). More concretely, the loss function can be expressed as $\mathcal{L} = \mathbb{E}[(y - Q(s,a))^2]$, where y denotes the value of the target. The target value can be expressed as $y = r + \gamma \max_{a'} Q(s', a')$, where γ denotes the discount rate.

As described, Q-learning is off-policy, as it tries to learn the greedy policy while using a different behavior policy for acting in the environment.

The loss is minimized using stochastic gradient descent. In order to make the network updates more stable, DQN leverages a technique known as *Experience Replay*. At each step of the data collection process, each single-step transition gets added to a replay buffer. During training, rather than simply using the latest data point to compute the loss, the algorithm samples a mini-batch of data from the replay buffer. By using uncorrelated samples in each batch and reusing data throughout training, this technique is both more stable and more sample efficient than vanilla Q-learning. To further improve stability, the parameters of the network θ used in the target estimation are fixed for multiple updates of the algorithm. This fixed *target network* therefore uses a different set of parameters θ^- than the parameters being updated, which mitigates potential instability stemming from the otherwise non-stationary targets. The update step can be succinctly expressed as

$$\theta \leftarrow \theta + \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-) - \hat{Q}(s, a; \theta))\nabla_\theta \hat{Q}(s, a; \theta),$$

where α is the learning rate. We include the detailed pseudo code for DQN in the appendix 1.

The baselines we compare to are the on-policy algorithms detailed above (i.e., PPO and NLPO). We leverage an off-the-shelf implementation of DQN [31] and modify it to perform language generation. Please note that deep RL algorithms are usually extremely difficult to debug and tune, as training is often unstable and takes a significant amount of time. Further more, adapting the codebase and integrating it within the RL4LM framework has proven to be a conceptually ambiguous and challenging task which requires substantial coding, troubleshooting, and debugging. Given the computational and time limits in this course, we consider getting a working bug-free version of DQN in RL4LM a success criterion for this part of the project.

5 Experiments

5.1 Data and evaluation method

We perform experiments on two separate NLG tasks: a synthetic task and an IMDB text continuation task.

Synthetic generation of increasing numbers. Due to compute constraints, and to test the implementation of our sampling techniques, we first choose a synthetic task. The dataset is a dummy dataset that contains prompts of only padding tokens. The task is to generate a sequence of increasing numbers. The reward and evaluation metric is the percentage of number pairs in ascending order in a sentence. We take each prompt length to be exactly 5, and for each rollout, the model generates 20 new tokens.

IMDB text continuation. With promising results on the synthetic task, we then run our methods on an IMDB positive text continuation task (Hugging Face data id: lvwerra/gpt2-imdb). The task is to complete a given snippet of a movie review as positively as possible. We take each prompt length to be exactly 5, and for each rollout, the model generates 40 new tokens. The output is evaluated using the *sentiment-analysis* pipeline in Hugging Face [32], which assigns a positiveness score between $[0, 1]$ to each sequence. As an additional evaluation metric for this task, we also examine the perplexity of the output.

5.2 Experimental details

We use the smallest version of GPT-2 [33] with 124M parameters as our common backbone pretrained model for all tasks, and then finetune each task using the downstream metrics detailed above. The GPT-2 model utilizes a transformers design and is pre-trained in a self-supervised approach on text data scraped from all outbound web page links on Reddit, excluding any Wikipedia pages. The tokenizer used is a variant of Byte Pair Encoding (BPE) that operates on individual bytes, and the size of the vocabulary is 50,257. The GPT-2 model used in the IMDB task is first fine-tuned on the IMDB dataset, then loaded in for further finetuning using RL.

Sampling methods. The model configuration for the synthetic task is outlined in Table 2. We use the Adam optimizer with minibatches of size 64. We use a learning rate of 0.00001. The PPO policy network parameters are updated 5 gradient steps per minibatch, and for the learned or delayed version, the masked policy is synced with the unmasked policy every 100 training steps. We train on this task for 450 training epochs.

We also perform a hyperparameter sweep for all sampling methods on the synthetic task. These values are outlined in the appendix in Table 3 in the appendix. Specifically, we sweep the values of μ for mirostat, p for typical and top-p sampling, η for η-sampling, and ϵ for ϵ-sampling.

The model configuration for the IMDB text generation task is outlined in Table 5 in the appendix. We use the Adam optimizer with minibatches of size 32. We use a learning rate of 0.00001. The PPO policy network parameters are updated 5 gradient steps per minibatch, and for the learned or delayed version, the masked policy is synced with the unmasked policy every 100 training steps. We train on this task for 450 training epochs.

DQN. Our goal for the DQN implementation is to get a working bug-free vanilla version of the algorithm for language generation tasks. Therefore, we leave the implementation and testing of combining DQN with sampling methods to future work, which we will discuss in detail later. The model configuration for DQN is outlined in Table 4 in the appendix. We use the Adam optimizer with minibatches of size 32, and set the learning rate to be 0.0001. We load the pretrained GPT-2

model as the value model, and add a multi-layer perceptron (MLP) module on top of the value model as the action-value head. The MLP takes as input the output hidden state of dimension 768 from the value model, and outputs an array of Q-values corresponding to the value of each token in the vocabulary given the hidden state. The MLP has three linear layers with a hidden size of 64, with ReLU activation functions in between. We include an illustration of the Q-network structure in Figure 4 in the appendix. The target Q-network shares the same structure as the Q-network, but is updated every 50 training steps. We set the replay buffer to be of size 1,500,000, essentially storing all possible transitions during training. The exploration policy, or behavior policy, is ϵ-greedy with ϵ annealed from 1 to 0.05 linearly over the first ten thousand steps and fixed at 0.05 afterwards. We train on the IMDB task for 35,000 steps.

5.3 Results

Figure 1: Comparison of delayed (or learned) masking on sampling methods

Sampling methods. Due to compute constraints we performed a hyperparameter sweep for each individual sampling method only on the synthetic generation of increasing numbers task. The results of the hyperparameter sweep for each method are reported in the appendix in Figures 6, 7, 8, and 9. We then used the best performing hyperparameter for the IMDB text continuation task. Figure 1 demonstrates that the delay masking strategy ("learned" strategy) significantly decreases the variance of the training process. Moreover, with the delay masking strategy, all algorithms (except Mirostat) obtain better training scores after 400 steps. It is clear that the learned strategy improves performance. Thus, we only consider the learned versions of the sampling algorithms moving forward.

Figure 2 shows the training performance of all the algorithms. For the sentiment score, we find that ϵ, η, and top-p all perform better than the baseline PPO algorithm with no truncation sampling. Notably, after 450 training steps, ϵ-sampling performs better than all other alternative methods, and also outperforms the NLPO baseline which uses top-p sampling. We also observe an unexpected behavior of our methods. Typical sampling and η-sampling are more sophisticated information-based extensions of top-p and ϵ-sampling, respectively, yet perhaps surprisingly, their simpler counterparts perform better on both tasks that we considered. Moreover, the most complicated method, Mirostat, performs the worst. We suspect that more tuning is needed to improve these algorithms, as they may be harder to optimize and more sensitive to hyperparameters.

For the perplexity score, we find that all methods result in higher perplexity scores than top-p, which overall performs well on both evaluation metrics. We note that more complicated methods such

as Typical, η, and Mirostat achieve larger perplexity scores than this baseline. An explanation for this is that these methods tune the parameters according to an entropy threshold that may result in generating sentences with higher perplexity. Notably, both ϵ and η sampling perform relatively well across both metrics (though ϵ sampling's high sentiment score does come at the expense of a high perplexity score).

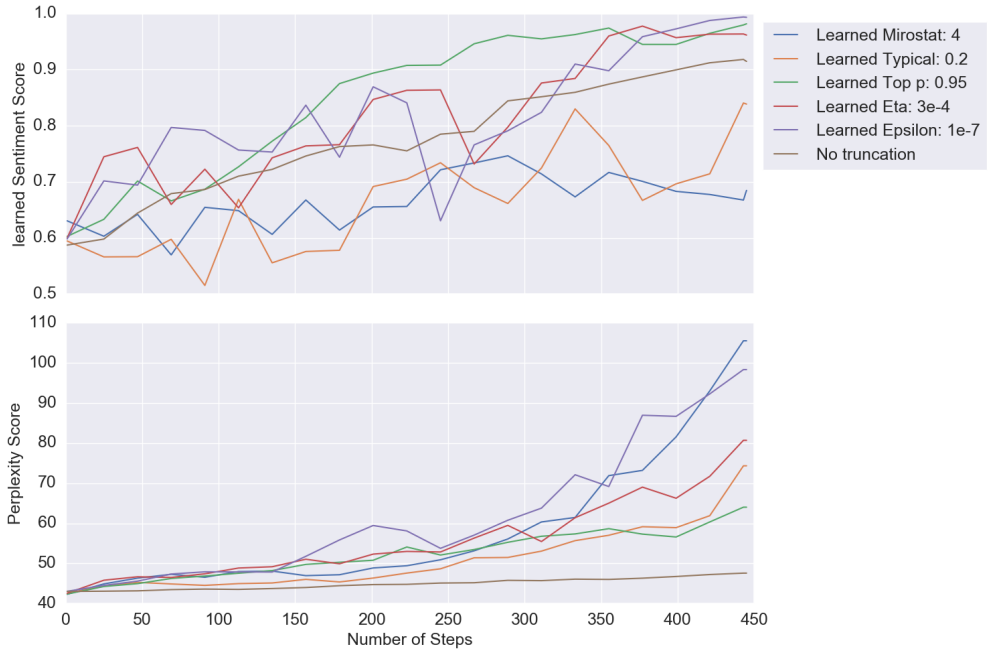

Figure 2: Performance of ϵ-sampling, η-sampling, typical sampling, NLPO, and PPO on the IMDB text generation task validation set

	Mirostat $\mu = 4$	**Typical** $p = 0.2$	**Top-p** $p = 0.95$	**Eta** $\eta = 3e^{-4}$	**Epsilon** $\epsilon = 1e^{-7}$	**No Truncation**
Perplexity	106.13	73.49	64.35	81.45	96.93	**46.75**
Sentiment Score	0.678	0.8446	0.982	0.962	**0.993**	0.919

Table 1: Out-of-sample test results after 450 steps

As shown in Table 1, we find that the relative performance of our algorithms on the training and validation data sets holds for the held out test set. ϵ sampling performs the best in terms of achieving the highest sentiment score, however it also achieves the second highest perplexity score. Perhaps unsurprisingly, no truncation results in the lowest perplexity score, as restricting the action space may come at the expense of producing plausible text. Our results suggest that alternative sampling techniques can be combined with PPO to effectively restrict the action space without compromising performance on certain metrics, however further work is required to quantify the tradeoff between sentiment score and perplexity for this particular task.

Here we show sample generations from each of the algorithms for a randomly picked prompt, which exemplify the trends explained above:

Prompt: I used to watch
Reference Text: <START-1>I used to watch this show when I was a little girl. When I think about it, I only remember it vaguely. If you ask me, it was a good show. Two things I remember vaguely are the opening sequence and theme song. In<END-1>
Mirostat: it regularly with the two-handed and the "G"-in-the-back-the-back-in-front-you-and-then-the-back-right-side-
Typical: this before seeing it, I know it did look similar to the other original and the other person got the real story of this great "Masters of the Past". The good about what they did is
Top p: movies in this movie I really recommend this movie for this site because I love this movie. If you're looking for a love of an epic adventure, this is an easy choice. This movie was a
Eta: them a long time ago, so I knew that I am truly a great, great, great, great grandmother, great-grandpa, great, great aunt-wife, great-grandmother,
Epsilon: a lot of films in the seventies and I saw this film in a high class and high school in New York. I was introduced with this wonderful plot development that is very well done. Unfortunately,
No Trucation: many films which attempt to depict America's most popular culture (in terms of cinema itself), but which in any case never fully captures or entertains. A truly incredible film in its own right, "

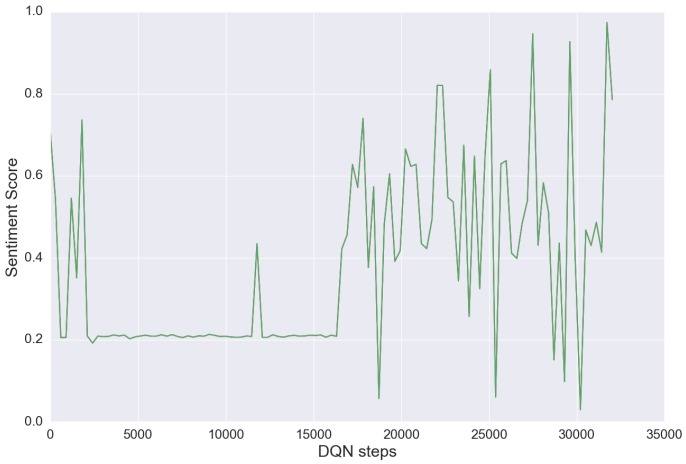

Figure 3: Performance DQN on the IMDB text generation task validation set

DQN. We successfully implemented vanilla DQN in the RL4LM framework, which is catered to language generation. The implementation and debugging turned out to be substantially more challenging than we initially believed. Thus, due to computational and time constraints, we could only run our DQN algorithm for approximately 30k steps. As shown in Figure 3, the algorithm is not able to learn much over the short course of training, and the initial performance is relatively noisy and unstable. We expect to potentially see convergence and improved performance by significantly extending the training time.

It is important to note that for every 4 gradient steps of the PPO algorithm, 128 new sentences need to be generated, which translates to $128 \times 40/4 = 1280$ new tokens, or equivalently $128/4 = 32$ new sentences generated per step. In contrast, every gradient update of DQN corresponds to only 1 newly generated token, or $1/40 = 0.025$ newly generated sentences. Moreover, our implementation of DQN only loads one GPT-2 model (as the value model), whereas PPO uses 2 GPT-2 models (one as the policy model, the other as the value model). For a fairer comparison, we could consider off-policy variants of actor-critic algorithms in the future.

DQN suffers from many well-known issues, which could partly explain why DQN performs so poorly on the IMDB task. These include:

1. **Sensitivity to hyperparameters**: DQNs can be more sensitive to the choice of hyperparameters, such as learning rate, discount factor, and network architecture. PPO is often considered more robust to hyperparameter choices. Since we are constrained by time and compute, we are not able to tune the hyperparameters of DQN.
2. **Slow convergence**: DQN usually takes a long time to converge. For the IMDB task this is problematic especially because the reward is noisy and sparse (reward is only given at the end of a sentence). Sparse rewards pose a challenge to TD-style updates and make it hard for Q-values to converge. Additionally, in DQNs, the target Q-values change during training, making the optimization problem non-stationary. This can also lead to instability and slow convergence.
3. **Correlated data**: as an off-policy algorithm, DQN reuses samples from the replay buffer, which consists of previously collected single-step transitions and rewards. Unlike PPO which samples independent trajectories, the training data for DQN exhibits complex dependency, which could lead to instability and slow convergence.
4. **Inefficient exploration**: the exploration strategy employed by DQN is naive dithering, namely ϵ-greedy. Exploration is only performed uniformly randomly with a small probability, which is highly inefficient in large state spaces (which are typical of NLG tasks). Indeed, from our inspection of generated texts from DQN, the tokens are often repeated or random (out-of-context).

The poor performance of DQN can also be explained by our specific implementation of the algorithm. First of all, we used a randomly initialized MLP to model the Q-function on top of the GPT-2 language model. The initial approximation of the Q-function would introduce errors in the estimates which can propagate and accumulate during training and destabilize learning. We have tried to adapt DQN to include a KL divergence reward (to penalize deviation away from GPT-2). However, our empirical results, shown in Figure 11 in the appendix, are not successful, possibly due to the relatively large scale of the KL term compared to the Q-function estimates. Another possible strategy to reduce error propagation is to freeze GPT-2 and train the MLP to learn an accurate representation of the Q-function before any fine tuning is performed.

Finally, we comment on the choice of architecture to implement DQN for NLG tasks. As discussed above, using a randomly initialized MLP head can cause our model to lose a crucial amount of representation power from the pretrained GPT-2 model. We suspect that this may be the reason why our outputs from DQN are often incoherent sentences with many repeated tokens. A solution for this is to implement an alternative architecture in the future where the output logits from GPT-2 are directly finetuned, treating them as surrogates for initial Q-values. Intuitively, the higher the Q-value, the more desirable the corresponding action. The same holds true for logits. Thus, the logits might be a good starting point for learning the Q-values.

6 Conclusion

Our work addresses two of the key empirical challenges faced by on-policy RL algorithms for natural language tasks: (1) training instability due to the large action space and (2) sample inefficiency. Due to time and compute constraints, as well as the inherent complexities of debugging and tuning deep RL algorithms, we were not able to complete all of the ambitious stretch goals that we outlined in our project proposal. Nonetheless, our preliminary results suggest that it is possible to leverage truncation sampling techniques to effectively restrict the action space without compromising performance on certain evaluation metrics, though further work is required to quantify the tradeoff between task preference metrics (such as sentiment score) and naturalness metrics (such as perplexity score). In addition, our implementation of a functioning DQN algorithm suggests that off-policy algorithms may be suitable for finetuning language models for downstream applications. Moving forward, potential future directions for further research include: comparing the performance of our sampling algorithms on a wide variety of NLG tasks (e.g. summarization, machine translation, question answering); additional tuning and training of our DQN algorithm; and combining both outputs of our work, i.e., incorporating our sampling techniques into DQN. Overall, we hope that our work can inform how RL can be used to improve language modeling and alignment, and provide a meaningful contribution to a fast-growing subfield with many potential applications to various downstream tasks.

References

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

[2] Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization. In *International Conference on Learning Representations, ICLR*, 2023.

[3] John Hewitt, Christopher D. Manning, and Percy Liang. Truncation sampling as language model desmoothing, 2022.

[4] Charlie Victor Snell, Ilya Kostrikov, Yi Su, Sherry Yang, and Sergey Levine. Offline RL for natural language generation with implicit language q learning. In *International Conference on Learning Representations, ICLR*, 2023.

[5] Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

[6] Samuel Kiegeland and Julia Kreutzer. Revisiting the weaknesses of reinforcement learning for neural machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1673–1681, Online, June 2021. Association for Computational Linguistics.

[7] Richard Yuanzhe Pang and He He. Text generation by learning from demonstrations. In *International Conference on Learning Representations*, 2021.

[8] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc., 2020.

[9] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*, 2018.

[10] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas, November 2016. Association for Computational Linguistics.

[11] Natasha Jaques, Judy Hanwen Shen, Asma Ghandeharioun, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Human-centric dialog training via offline reinforcement learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3985–4003, Online, November 2020. Association for Computational Linguistics.

[12] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[13] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi (Eric) Yuan. Interactive fiction games: A colossal adventure. In *AAAI 2020*, October 2019. ArXiv.

[14] Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. Recursively summarizing books with human feedback, 2021.

[15] Reiichiro Nakano, Jacob Hilton, S. Arun Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332, 2021.

[16] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[17] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR*, 2016.

[18] Alice Martin, Guillaume Quispe, Charles Ollion, Sylvain Le Corff, Florian Strub, and Olivier Pietquin. Learning natural language generation with truncated reinforcement learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 12–37, Seattle, United States, July 2022. Association for Computational Linguistics.

[19] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. In *International Conference on Learning Representations, ICLR*, 2017.

[20] Prithviraj Ammanabrolu and Matthew J. Hausknecht. Graph constrained reinforcement learning for natural language action spaces. *ArXiv*, abs/2001.08837, 2020.

[21] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[22] Prithviraj Ammanabrolu, Liwei Jiang, Maarten Sap, Hannaneh Hajishirzi, and Yejin Choi. Aligning to social norms and values in interactive narratives. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5994–6017, Seattle, United States, July 2022. Association for Computational Linguistics.

[23] Siddharth Verma, Justin Fu, Mengjiao Yang, and Sergey Levine. Chai: A chatbot ai for task-oriented dialogue with offline reinforcement learning, 2022.

[24] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

[25] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[26] Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally typical sampling, 2022.

[27] Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R. Varshney. MIROSTAT: A neural text decoding algorithm that directly controls perplexity. In *International Conference on Learning Representations*, 2021.

[28] Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. A modular rl library to fine-tune language models to human preferences. `https://github.com/allenai/RL4LMs`, 2023.

[29] Leandro von Werra, Jonathan Tow, Shahbuland Matiana, reciprocated, Alex Havrilla, cat state, Louis Castricato, Alan, Ayush Thakur, Alexey Bukhtiyarov, Duy V. Phung, aaronrmm, Fabrizio Milo, Daniel, Dong Shin, Ethan Kim, Justin Wei, Manuel Romero, Nicky Pochinkov, Omar Sanseviero, Reshinth Adithyan, Sherman, Thomas Simonini, Vladimir Blagojevic, Zack Witten, crumb, marcobellagente93, smellslikeml, and thomfoster. CarperAI/trlx: v0.5.0: Initial NeMo integration, HH example, and improved Hugging Face integration, February 2023.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[31] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[32] Huggingface pipelines. `https://huggingface.co/docs/transformers/main_classes/pipelines`.

[33] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

A Appendix

Algorithm 1 Deep-Q Learning pseudo code

Input C, α, $D = \{\}$, `maxIter`
Initialize θ, $\theta^- = \theta$, $t = 0$
while $t <$ `maxIter` **do**
  Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \theta)$
  Observe reward r_t and next state s_{t+1}
  Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
  Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
  for j in minibatch **do**
    if episode terminated at step $i + 1$ **then**
      $y_i = r_i$
    else
      $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \theta^-)$
    end if
    SGD on $(y_i - \hat{Q}(s, a))^2$ for parameters θ, where $\delta\theta = \alpha(y_i - \hat{Q}(s_i, a_i; \theta))\nabla_\theta \hat{Q} s_i, a_i; \theta)$
  end for
  $t = t + 1$
  if $\text{mod}(t, C) == 0$ **then**
    $\theta^- \leftarrow \theta$
  end if
end while

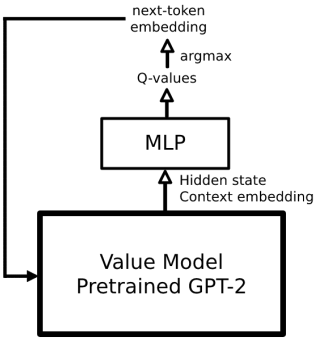

Figure 4: GPT-2 with an MLP head as a Q-network

Model Params	value
On-policy agent	batch size: 64 learning rate: 0.00001 steps per update: 1280 total number of steps: 256000 epochs per update: 5 discount factor: 0.99 gae lambda: 0.9 clip ratio: 0.2 value function coeff: 0.1 target update iterations: 100 for NLPO/top-p, 1 for others
Off-policy agent	batch size: 5 learning rate: 0.0001 replay buffer size: 1500000 warm start: 20 polyak target soft update rate: 0.005 discount factor: 1 exploration rate: annealed linearly from 1 to 0.05
decoding	sampling: true top k: null min length: 20 max new tokens: 20
tokenizer	padding side: left truncation side: left max length: 5

Table 2: Model configuration for each method on the synthetic task

Method	Hyperparameters	Best hyperparameter value
top-p	$p \in \{0.89, 0.90, 0.92, 0.95, 0.99\}$	0.95
typical	$p \in \{0.2, 0.9, 0.92, 0.95\}$	0.2
ϵ	$\epsilon \in \{0.0001, 0.00001, 0.000001, 0.0000001\}$	0.0000001
η	$\eta \in \{0.004, 0.002, 0.0009, 0.0006, 0.0003\}$	0.0003

Table 3: Hyperparameter sweep for each method on the synthetic task

Model Params	value
batch size:	32
learning rate:	0.0001
discount factor (γ):	1
target network update interval (C):	50
initial exploration fraction (ϵ):	1.0
final exploration fraction (ϵ):	0.05

Table 4: Model configuration of DQN

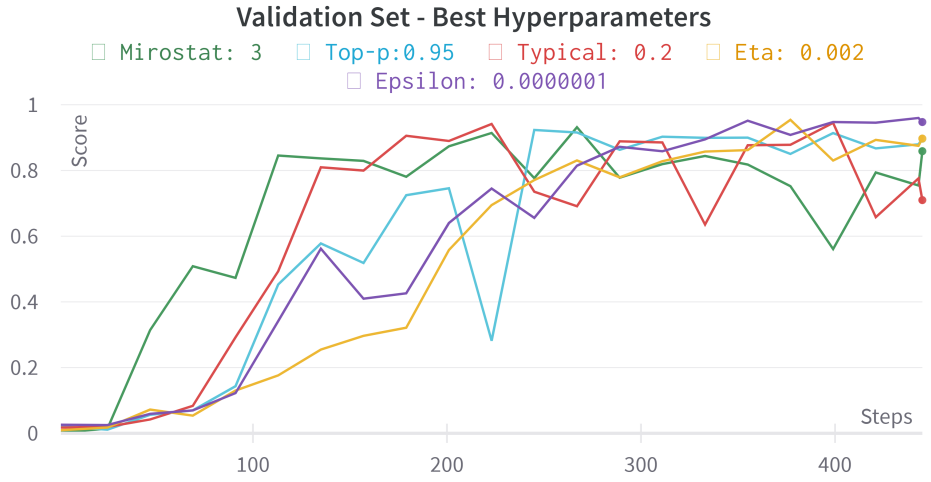

Figure 5: Performance of all methods with the best hyperparameter

Model Params	**value**
On-policy agent	batch size: 32 learning rate: 0.0001 steps per update: 128 total number of steps: 12800 epochs per update: 4 discount factor: 1 gae lambda: 0.95 clip ratio: 0.2 value function coeff: 1 target update iterations: 1
Off-policy agent	batch size: 5 learning rate: 0.001 replay buffer size: 1500000 warm start: 20 polyak target soft update rate: 0.005 discount factor: 1 exploration rate: annealed linearly from 1 to 0.05
decoding	sampling: true top k: null min length: 0 max new tokens: 40
tokenizer	padding side: left truncation side: right max length: 984

Table 5: Model configuration for each method on the IMDB task

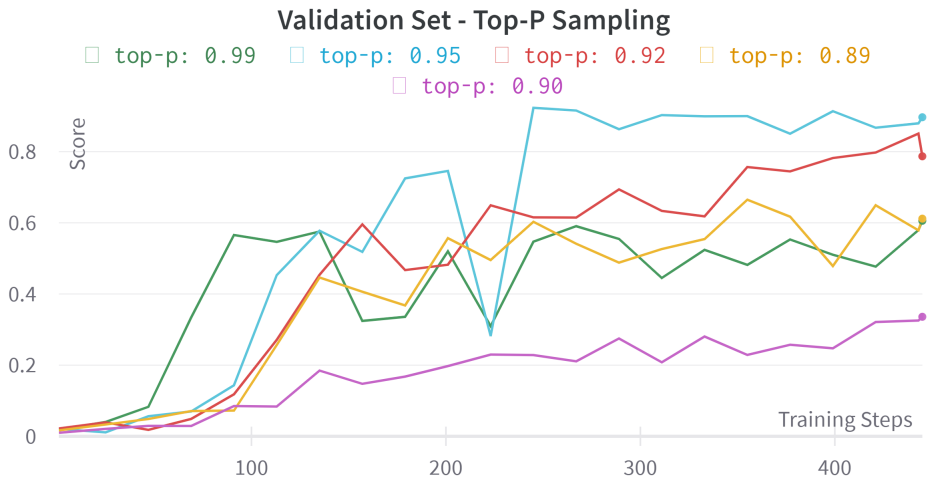

Figure 6: Hyperparameter sweep on NLPO (using top-p sampling)

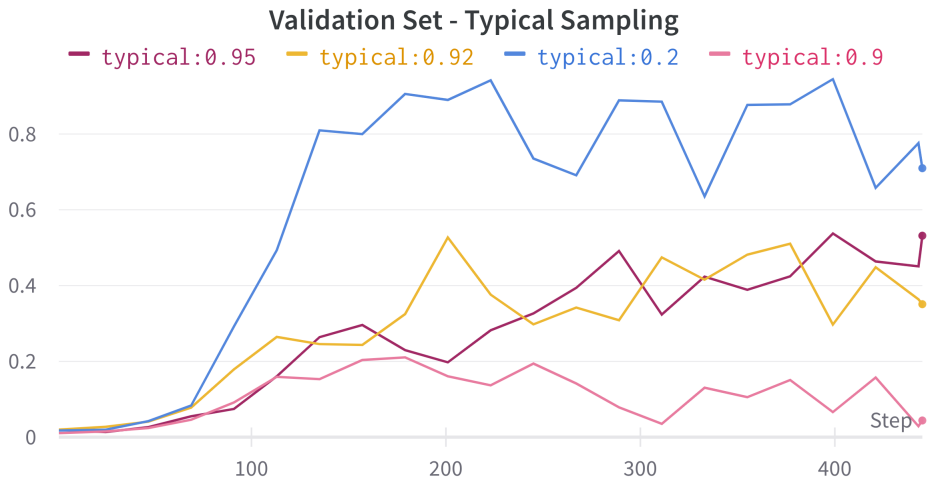

Figure 7: Hyperparameter sweep on typical sampling

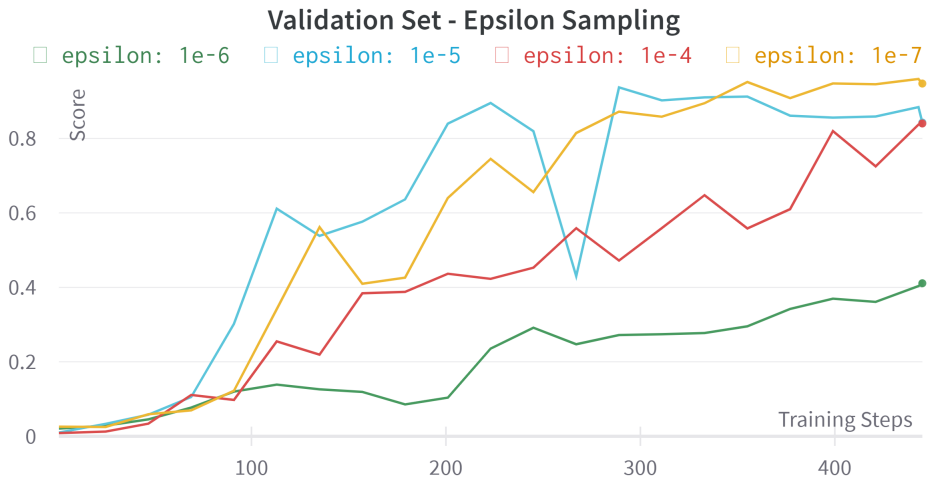

Figure 8: Hyperparameter sweep on ϵ-sampling

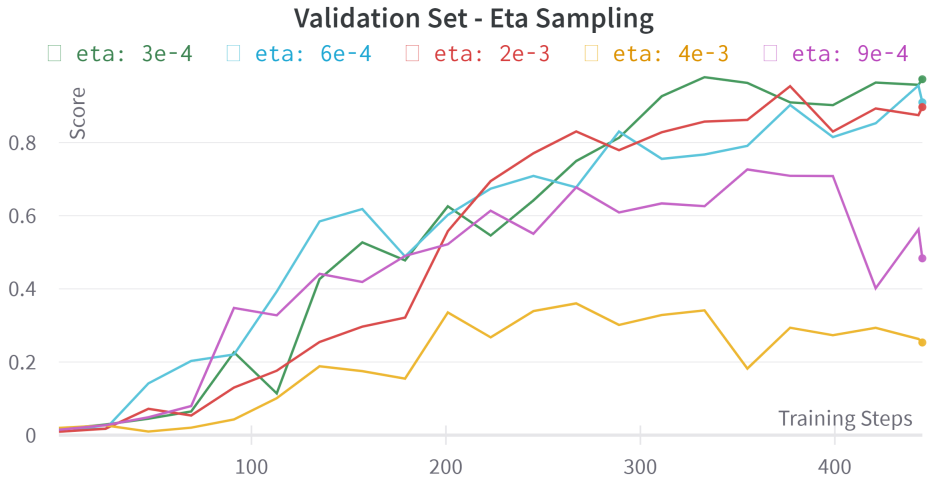

Figure 9: Hyperparameter sweep on η-sampling

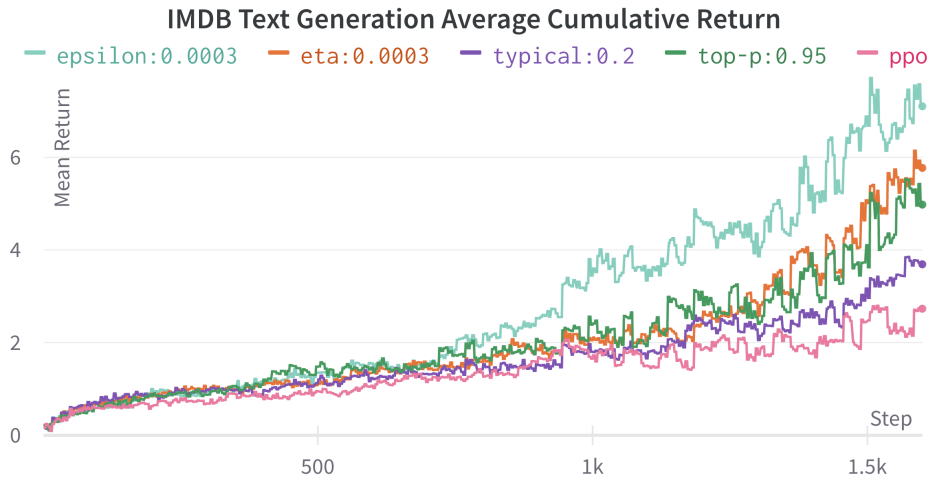

Figure 10: Performance of ϵ-sampling, η-sampling, typical sampling, NLPO, and PPO on the IMDB text generation task (from the TRLX repo)

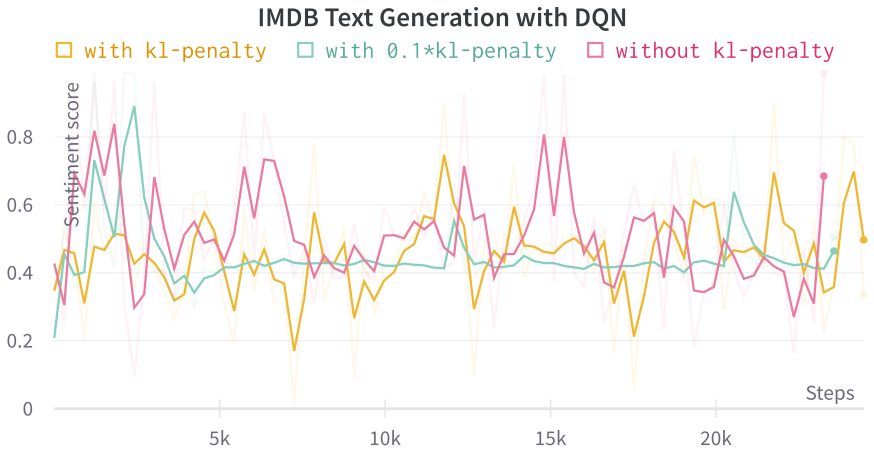

Figure 11: Performance of DQN on the IMDB text generation task with KL regularization