Documentation of Bit Sequence mini RL project

Chau Yu Hei*
HKUST
20644747
yhchau@connect.ust.hk

Abstract

GFlowNets provide a new paradigm for finetuning LLMs with respect to some energy function. However, a common problem is that strings with high probability are often very rare compared to the strings with low probability. How do we efficiently tune LLMs using the GFlowNet algorithm so that it is able to learn from sparse rewards? In this mini-project, we attempt to tackle the problem of generating a randomly prechosen bit string using old RL methods.

1 Background

Recent work has shown that GFlowNets are able to generate diverse samples for LLMs, and finetune LLMs according to Bayesian updates (Hu et al. [2024]). However, their experiments are limited to generating text sequences of short length. How do we generate text sequences of larger lengths? It is evident that most strings have low reward due to the large search space of the set of all possible text sequences.

In this mini-project, we attempt to tackle a similar toy problem, that is, to generate a specific bit sequence in the set of all bit sequences. We first apply a naive DQN method for sequence generation, and show that the method only works for small n. Then we attempt to apply Hindsight Experience Replay (HER) (Andrychowicz et al. [2017]) by OpenAI to see whether the method successfully generates the correct bit sequences for longer sequence lengths.

2 Problem setting

Here we attempt to use a DQN to generate a predesignated randomly picked target bit sequence. In each setting, an integer $1 \leq n \leq 50$ is fixed, and the state space is $\mathcal{S}_n = \{0,1\}^n$. If we make the graph complete by allowing full 2^n actions for every transition, the DQN will be too expensive in memory to parameterize (since we need to compute $\max$ over the action space). So we consider the actions to be flipping a bit $\mathcal{A}_n = \{i \in \mathbb{Z} | 0 \leq i \leq n-1\}$ (using computer indexing convention).

The goal is to find a policy $\pi : (s, g) \in \mathcal{S}_n \times \mathcal{S}_n \mapsto \pi(s, g) \in \mathcal{A}_n$ such that the agent is able to start from s and arrive at the goal $g \in \mathcal{S}_n$ in the least amount of steps. To this end, we train a goal conditioned DQN $(s, g, a) \mapsto Q_g(s, a) \in \mathbb{R}$. The reward function $R(s, g, a) = 1$ iff the next state $s \to_a s' = g$, and would be 0 otherwise.

The intuition is that, for low values of n (maybe $1 \leq n \leq 10$?), a naive DQN is able to learn to arrive at the bit sequence due to exhausting the search space (along with using the parallel processing by GPUs). However, for large values of n, this would be infeasible, since the reward is too sparse.

*Currently not a student yet :(

Not in the 38th Conference on Neural Information Processing Systems (NeurIPS 2024)

Table 1: DQN experiments (max 3000 episodes)

	$n =$						
	5	10	15	20	30	40	50
HER-UVFA	✓	✓	✓	✗	?	?	?
Vanilla-UVFA	✓	✓	✗	✗	?	?	?
HER-HAR	?	?	?	✓	✓	✓	✓
Vanilla-HAR	?	?	?	✓	✗	✗	✗

✓: Converged to 1.00 success rate
✗: Failed to converge to 1.00 success rate
?: Didn't run

3 Experiments

In the below experiments, we run vanilla-DQN and HER-DQN with $n = 5, 10, 15, 20, 30, 40, 50$. We test two kinds of network architectures, namely UVFA (Schaul et al. [2015]), and a Handcrafted ARchitecture (HAR) that has strong inductive bias towards the bitflipping state-action space. Please refer to the Appendix 4 for more details on the network architectures.

The agent will perform at most n steps in each episode. For all experiments, we run 16 agents simultaneously for training episodes, while 1024 agents will be run simultaneously for testing episodes. We only modify the replay buffer during the training episodes, while the testing episodes do not alter the replay buffer. The replay buffer has a maximum 2^{15} capacity, and previous experiences will be sampled from the buffer uniformly during replay (optimizing Q network). All other relevant hyperparameters (such as lr, γ, etc) are kept the same for each experiment.

The starting state $s \in \mathcal{S}_n$ and target $g \in \mathcal{S}_n$ for each 16 agents (resp. 1024) are initialized randomly and uniformly. The agents follow a ϵ-probability random exploration chance starting from $\epsilon = 1.0$ and decaying to $\epsilon = 0.1$ with exponential decay factor 0.999. To save time, we run at most 3000 episodes for each experiment. Assuming the states explored are uniformly random, we expect on average approximately $16 \times n \times 3000 = 48000n$ states to be explored during training, which is dominated by $|\mathcal{S}_n| = 2^n$ for large n.

As expected, vanilla DQN with UVFA is unable to learn the optimal policy for $n = 15, 20$ since the probability of getting the reward is too sparse ($\frac{48000n}{2^n} << 1$). The DQN mostly sees 0 all the time as the reward, and as a result unable to learn anything useful. If we use the handcrafted architecture (HAR), vanilla DQN is able to learn for $n = 20$ due to strong inductive bias towards the bitflipping problem. However, it fails for $n = 30, 40, 50$ (Table 1).

In contrast, HER-DQN with UVFA succeeds on all $n = 5, 10, 15$, and with HAR succeeds on all $n = 20, 30, 40, 50$. Although HER-DQN fails for $n = 20$ (UVFA), the success rate and mean distance to target are improving much more rapidly as compared to vanilla-DQN with UVFA on $n = 20$ (see 4.1). This shows the superiority of HER-DQN.

By experimenting with a handcrafted architecture with strong inductive bias towards bitflipping state-action space, we dispel the possibility that an inappropriate architecture or optimization difficulty causes vanilla-DQN to fail on high n, the only reason that could be is that vanilla-DQN is prone to sparse rewards.

For detailed code on the experiments, please refer to the Github repo. For more details on the experiment setup, and some miscellaneous challenges faced and tricks used, please refer to the appendix. For other visualizations and graphs, please refer to Vanilla-DQN notebook HER-DQN notebook in GitHub. For further discussions and research motivated by this, please refer to (private documentation) or email.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay.

Advances in neural information processing systems, 30, 2017.

Edward J. Hu, Moksh Jain, Eric Elmoznino, Younesse Kaddar, Guillaume Lajoie, Yoshua Bengio, and Nikolay Malkin. Amortizing intractable inference in large language models. In *Proceedings of the International Conference on Learning Representations*, 2024.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1312–1320. JMLR.org, 2015. URL `http://proceedings.mlr.press/v37/schaul15.html`.

4 Appendix

4.1 Details for $n = 20$

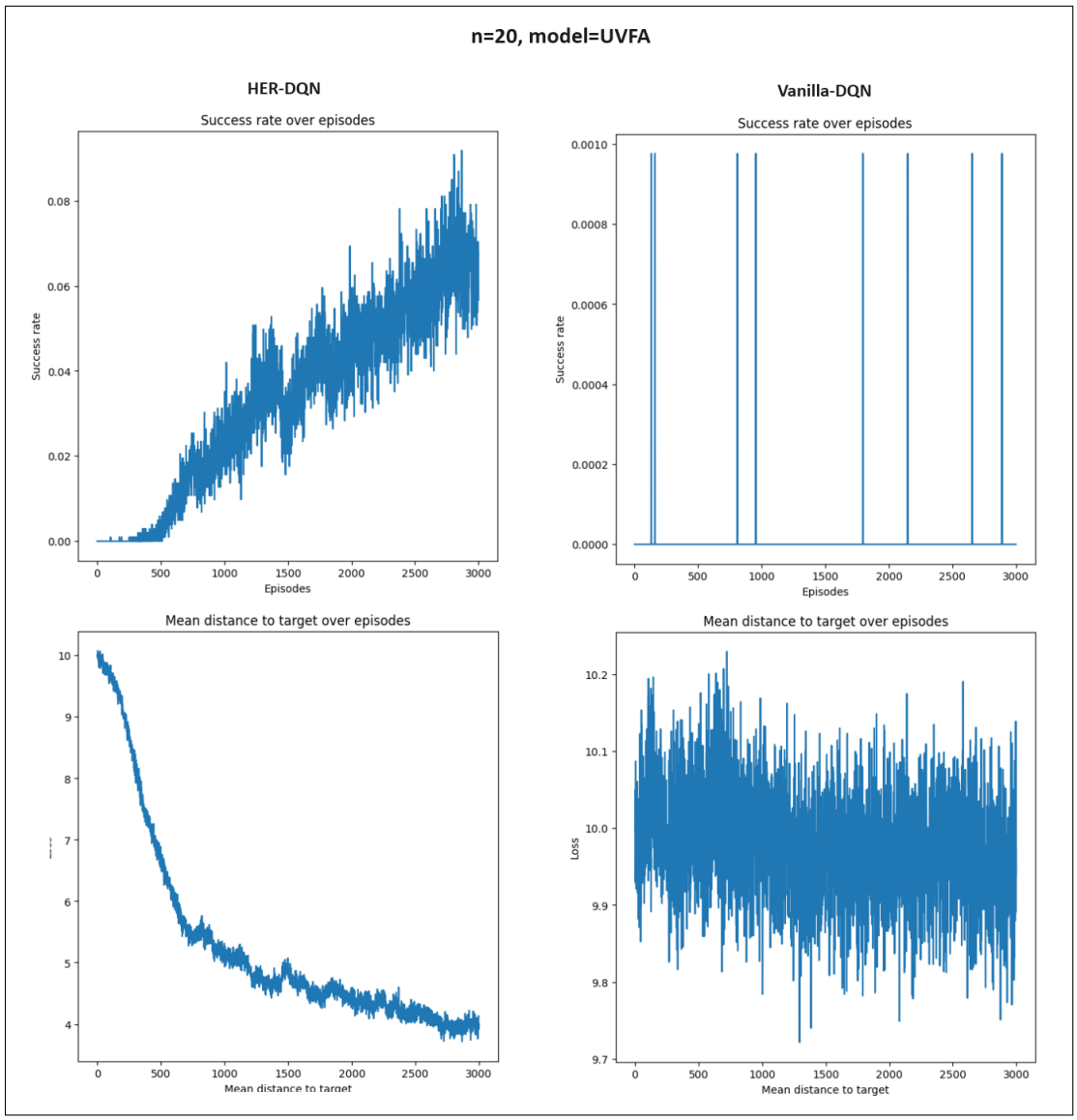

Figure 1: Metrics for $n = 20$

4.2 Network architectures

For UVFA Schaul et al. [2015], the authors recommend learning the states and goals in the same latent space and using a simple dot product as the Q-value. It is well-known that pre-composing linear transforms before dot product is equivalent to a bilinear operation. Therefore, we simply use a single nn.Linear layer following by GELU activation to map the state and goal to $\mathbb{R}^{128}$ vectors, and use a learnable bilinear form $\mathbb{R}^{128} \otimes \mathbb{R}^{128} \to \mathbb{R}^n$.

For the Handcrafted ARchitecture (HAR), it is intuitive that whether to flip the bit at i depends only on $\|s - g\|_1$ and $|s[i] - g[i]|$. We therefore parameterize $Q_g(s, a) = \phi_\theta(\|s - g\|_1, |s[a] - g[a]|)$ where $\phi_\theta : \mathbb{R}^2 \to \mathbb{R}$ is a neural network with parameters θ. Formally, we can define an equivalence

relation $\sim$ on $\mathcal{S}_n \times \mathcal{S}_n$ invariant to permutation of the bits, and note that the optimal $Q^*(s, g, a)$ factors through $\sim$. So this is like acting on the quotient space of the original state-goal-action space (which has less elements).

4.3 Poylak averaging trick

Initially, I tried to fit the DQN directly using Bellman's equation. However, the training loss is unstable and blows up occasionally, which makes the training dynamics quite noisy. After a lot of debugging, I searched online and usually people use Polyak averaging trick to separate the policy network and target network, where the target network's weights are updated slower than the policy network weights by exponential averaging.

Let θ, ϕ be the weights of the Q_{policy} and Q_{target}. Initially, we set $\theta = \phi$. During optimization, we compute $\theta \leftarrow \theta + lr * \frac{dL}{d\theta}$, and afterwards update $\phi \leftarrow 0.99\phi + 0.01\theta$. So Q_ϕ will update slower compared to Q_θ. Note that the actual update step for θ is the Adam optimizer algorithm instead of simple SGD.

4.4 Target clipping trick

After solving the noisy updates problem, another problem emerged where the loss somehow keeps increasing for large n (irrespective of network architecture and whether HER is used). Note that this is worse when HER is present. After some debugging, the source of this problem is due to the update $Q_{policy}(s, g, a) = R_g(s, a) + \gamma \max_a Q_{target}(s, g, a)$. In my experiments, I picked a relatively large γ close to 1 (or else the rewards will be too small when $n = 50$).

Assuming that $Q_{policy}(s, g, a), Q_{target}(s, g, a)$ approximately follows the same distribution over the space $\mathcal{S}_n \times \mathcal{S}_n \times \mathcal{A}_n$ (since the target is an averaged version of policy), we intuitively have $E_{s,g,a}[Q_{policy}(s, g, a)] < E_{s,g}[\max_a Q_{target}(s, g, a)]$ since there is maximum in the target. If the increase is larger than γ in the optimization trajectory $\theta_1, \theta_2, \theta_3, \cdots$, then it is expected that the Q-values explode to infinity.

This also explains why this phenomenon is much worse when HER is present. With HER, much more saved experiences have $R_g(s, a) = 1$ (since HER biases the distribution in the replay buffer to having more rewards with non-zero values).

To tackle this problem, we note that the actual target for an optimal Q^* is in $[0, 1]$, since the termination states are at $s = g$, which gives reward 1, and other states have

$$Q^*(s, g, a) = \underbrace{R_g(s, a)}_{=0} + \gamma \underbrace{\max_a Q^*(s, g, a)}_{\leq 1}$$

by induction. By clipping the target values in $[0, 1]$, the training loss is stabilized.