An Embarrassingly Parallel Mixture of GFlowNets

Vincent Quirion
Université de Montréal

Moksh Jain
Université de Montréal

Emmanuel Bengio
Recursion

Jason Hartford
Recursion

Yoshua Bengio
Université de Montréal

Abstract

Generative flow networks (GFlowNets) are a powerful class of methods for sampling from a distribution defined by an energy function. We develop an embarrassingly parallel method to parallelize GFlowNets across a large number of compute nodes by using clustering to partition the space of terminal states, and then training a mixture of GFlowNets in parallel, each on a distinct part. We leverage the fact that, once trained, GFlowNets give an estimate of the normalizing constant. This lets us draw a sample from the mixture, by first selecting a GFlowNet with probability proportional to its estimated normalizing constant and then running the selected GFlowNet's policy to draw a sample. We show that this simple procedure produces (asymptotically) unbiased samples from the target distribution, and evaluate the procedure on a synthetic problem and a molecule generation problem. Empirically, we found that the mixture converges to distributions that produce more diverse solutions than a single GFlowNet trained with an equivalent amount of compute, and because of the benefits of parallelism, we were able to train the mixture with a significant saving in wall-clock time.

1 Introduction

Will be in final paper

2 Background

2.1 Generative Flow Networks

Consider a directed acyclic graph $G = (\mathcal{S}, \mathbb{A})$, with state space $\mathcal{S}$ and action space $\mathbb{A}$ (space of transitions between two states $s_t \rightarrow s_{t+1}$). Let $s_0 \in \mathcal{S}$ be the initial state, the unique state with no incoming edges, and $\mathcal{X} \subseteq \mathcal{S}$ be a set of terminating states (states without outgoing edges) representing objects. Given an unnormalized, nontrivial and nonnegative reward function $R : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ over the set of terminating states, the goal of a GFlowNet model is to learn a stochastic generative policy $\pi(a_t|s_t)$ that samples objects $x \in \mathcal{X}$ such that $P_\pi(x) \propto R(x)$. A GFlowNet constructs an object by following a sequence of transitions from the initial state to a terminating state (*complete trajectory*) $\tau = (s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_n)$, where $\tau \in \mathcal{T}$, the set of all possible complete trajectories.

As in Bengio et al. [2021], we define a trajectory flow $F : \mathcal{T} \rightarrow \mathbb{R}_{\geq 0}$, from which we derive the state flow $F(s)$ and edge flow $F(s_t \rightarrow s_{t+1})$:

$$F(s) = \sum_{\tau \ni s} F(\tau), \qquad F(s_t \rightarrow s_{t+1}) = \sum_{\tau \ni s_t \rightarrow s_{t+1}} F(\tau). \tag{1}$$

A nontrivial (not identically zero) trajectory flow F induces a probability P over the set of complete trajectories (Malkin et al. [2022]):

$$P(\tau) = \frac{F(\tau)}{Z}, \;\; Z = F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau) = \sum_{x \in \mathcal{X}} F(x). \tag{2}$$

Hence, we define the corresponding forward policy $P_F(s_{t+1}|s_t)$ and backward policy $P_B(s_t|s_{t+1})$:

$$P_F(s_{t+1}|s) = \frac{F(s_t \to s_{t+1})}{F(s_t)}, \;\; P_B(s_t|s_{t+1}) = \frac{F(s_t \to s_{t+1})}{F(s_{t+1})}. \tag{3}$$

The trajectory flow also brings about action distributions $P_F(-|s)$ over the children of each nonterminal state s, which allow for the factorization of P as

$$P(\tau = (s_0 \to s_1 \to ... \to s_n)) = \prod_{t=1}^{n} P_F(s_t|s_{t-1}). \tag{4}$$

Correspondingly, each noninitial state s has a distribution $P_B(-|s)$ over its parents such that for each terminal state x,

$$P(\tau = (s_0 \to s_1 \to ... \to s_n)|s_n = x) = \prod_{t=1}^{n} P(s_{t-1}|s_t). \tag{5}$$

Intuitively, these flow values can be thought of as the amount of water flowing through the edges representing transitions (like pipes) and the nodes representing states (like tees, connecting pipes) (Malkin et al. [2022]), with $R(x)$ being the amount of water that flows out of a terminal state x.

Given a space of objects $\mathcal{X}$ and a nontrivial non negative reward function $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$, the goal of a GFlowNet is to learn a stochastic generative policy π that samples objects $x \in \mathcal{X}$ proportionally to $R(x)$, that is, $P_\pi(x) \propto P(x)$. Consider a directed acyclic graph (DAG) $G = (\mathcal{S}, \mathbb{A})$ with state space $\mathcal{S}$ and action space $\mathbb{A}$. Let $s_0 \in \mathcal{S}$ be the initial (source) state, the only state with no incoming edges

2.2 Trajectory balance

When training with *trajectory balance* (Malkin et al. [2022]), the *de facto* learning algorithm for GFlowNets, the model learns three estimators P_F, P_B, and Z.
(Explain more for final paper?)

3 Mixture of GFlowNets

Consider a standard GFlowNet setting, where there is some terminal state space $\mathcal{X}$, from which we want to sample objects proportionally to a reward function $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$. Using a standard GFlowNet, we can train a policy π for which $P_\pi(x) = \frac{R(x)}{\sum_{x' \in \mathcal{X}} R(x')}, \; x \in \mathcal{X}$.

We propose to make the problem of learning a GFlowNet policy π embarrassingly parallel by training a mixture of completely independent subordinate policies $\{\pi_1, \pi_2, ..., \pi_k\}$, each targeting a specific part of the space of terminal states $\mathcal{X}$.

Training the mixture Let $F_p(\mathcal{X}, k)$ be a partition function that partitions $\mathcal{X}$ into disjoints subsets $\mathcal{X}_i$ such that $\mathcal{X} = \cup_{i \in \mathcal{I}} \mathcal{X}_i$ for some index set $\mathcal{I} = \{1, 2, \ldots, k\}$, and $\mathcal{X}_j \cap \mathcal{X}_i = \emptyset$ for all $i, j \in \mathcal{I}$. Given such a partition, we can define alternate reward functions $R_i(x) = R(x)\mathbf{1}(x \in \mathcal{X}_i)$ and use them to train k separate policies $\{\pi_1, \pi_2, ..., \pi_k\}$ with trajectory balance such that, at convergence, $P_{\pi_i}(x) = \frac{R(x)\mathbf{1}(x \in \mathcal{X}_i)}{R(\mathcal{X}_i)}$[1].

A potential problem of using reward functions defined as such is they can be rather sparse, which makes it difficult for a subordinate policy π_i to converge. To circumvent this problem, we can include in each batch trajectories that lead to off-policy samples that we know are in $\mathcal{X}_i$ and for which $R_i(x) > 0$.

[1]To simplify notation, we overload the reward function to define a measure, $R(\mathcal{X}) := \sum_{x' \in \mathcal{X}} R(x')$, such that $P_\pi = \frac{R(x)}{R(\mathcal{X})}$.

Sampling from the mixture First, we use the normalizing constants $\{Z_1, Z_2, ..., Z_k\}$ learned by the subordinated policies during their training with trajectory balance to obtain approximations of $R(\mathcal{X}_i) \approx Z_i$ and $R(\mathcal{X}) \approx \sum_{i \in \mathcal{I}} Z_i$. To sample from the mixed policy π, we sample from its subordinate policies $\{\pi_1, \pi_2, ..., \pi_k\}$ proportionally to their estimated $R(\mathcal{X}_i)$, such that $P_\pi(x) = \sum_{i \in \mathcal{I}} (\frac{R(\mathcal{X}_i)}{R(\mathcal{X})} \times \frac{R_i(x)}{R(\mathcal{X}_i)}) = \frac{1}{R(\mathcal{X})} \sum_{i \in \mathcal{I}} R(x) = \frac{R(x)}{R(\mathcal{X})}$ when the subsets are disjoints.

The complete training and sampling procedures are depicted in Algorithm 1.

Algorithm 1 Mixture of GFlowNets.

```
1:  Use the partitioning function $F_p(\mathcal{X}, k)$ to split the space of terminal states $\mathcal{X}$ into $k$ disjoint
    subsets $\{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_k\}$.

2:  // Train the $k$ subordinate policies in parallel
3:  for $i \in \mathcal{I} = \{1, 2, ..., k\}$, in parallel do
4:      Define an alternate reward function $R_i(x) = R(x)\mathbf{1}(x \in \mathcal{X}_i)$
5:      Initialize the subordinate forward and backward policies $P_{F\,i}$, $P_{B\,i}$, and learnable subordinate
        normalizing constant $Z_i$.
6:      for each training step $t = 1$ to $T$ do
7:          Collect a training batch of $B \times (1-$off_policy_ratio$)$ trajectories based on the subordinate
            forward policy $P_{F\,i}$
8:          Compute the reward $R_i(x)$ for each on-policy samples in the batch
9:          Add to the training batch $B \times$ (off_policy_ratio) trajectories that lead to samples for
            which $R_i(x)$ is known and $R_i(x) > 0$
10:         Update the subordinate GFlowNet model according to the trajectory balance loss
11:     end for
12: end for

13: // Sampling from the mixed policy
14: Compute the sum of the subordinate normalizing constants $\sum_{i \in \mathcal{I}} Z_i$
15: for each $i = 1$ to $N$ do
16:     Choose a subordinate policy $\pi_i$ from $\{\pi_1, \pi_2, ..., \pi_k\}$ at random with probability
        $\frac{Z_i}{\sum_{i \in \mathcal{I}} Z_i} \approx \frac{R(\mathcal{X}_i)}{R(\mathcal{X})}$
17:     Sample from $\pi_i$ as with a standard GFlowNet
18: end for
```

4 Experiments

4.1 Hypergrid environment

4.1.1 Setup

We first study the performances of the Mixture method in the parameterized hypergrid environment introduced in **?**. In this environment, the objects sampled are cells of a D-dimensional hypercubic grid of height H. The agent starts at coordinate $x = (0, 0, ..., 0)$, and can move around the grid only by taking action a_i, which increases coordinate i by 1 (actions that would increase i past $H - 1$ are masked). A trajectory ends once the agent takes the *stop* action. The environment's reward function has a mode near each corner of the hypergrid, 2^D in total, which are surrounded by plateaux of moderate reward and separated by troughs of base reward R_0 (Figure 1). As R_0 decreases, exploration in the environment becomes harder.

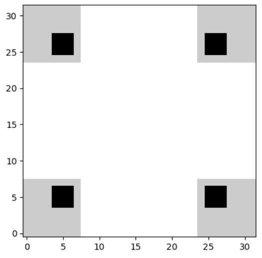

Figure 1: Target distribution of a 32x32 hypergrid

The baseline results were computed with an open source implementation of GFlowNets in the hypergrid[2], which we then extended with an

[2]https://github.com/saleml/gfn

implementation of the mixture method to evaluate its performances in the environment.

Optimal partitioning We define an *optimal partition* as a partition for which each subset of the space of terminal state contains an equal amount of probability mass, i.e., $R(\mathcal{X}_i) \approx R(\mathcal{X}_j)$, for all $i, j \in \mathcal{I}$. Optimal partitions are desirable since they allow to not have to rely on the learned subordinate normalizing constants during the sampling process. In fact, given a mixed policy π of subordinate policies trained on parts of an optimal partition, we can sample from π by uniformly sampling one of its suboordinate policies, and sampling from it. For every configuration of the hypergrid $\langle D, H, ..., R_0, ...\rangle$, we can define an optimal (*perfect*) partition of 2^D parts, such that each part i contains a single mode m and the points for which m is the nearest mode. We include these results since an improved version of the method could include learning an optimal partition over the course of training, which would lead to an asymptotically perfect partition of the state space.

Random partitioning Since knowing an optimal partition of $\mathcal{X}$ *a priori* is impossible for most real-world environments and efficiently learning an optimal partition is still out of reach, we also test the method with a random partition function. Assume we have a feature vector, x, that describes each state in $\mathbb{R}^d$; we project each feature vector onto the d-dimensional unit sphere by normalizing the feature, $\tilde{x} = x/|x|$, and then we randomly partition the sphere into k classes with a randomly initialized linear classifier with bias term equal to the mean so that the decision boundaries are through the center (Figure 2).

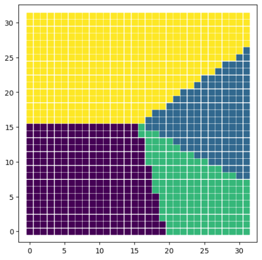

Figure 2: Example of a random partition of a 32x32 hypergrid

Experimental details We run the simulations in small ($H = 8$), medium ($H = 16$) and large ($H = 32$) hypergrids of 2 and 3 dimensions. We test on both the easy ($R_0 = 0.1$) and harder ($R_0 = 0.0001$) variants of these grids. For each configuration $\langle D, H, ..., R_0, ...\rangle$, we evaluate a standard GFlowNet (**?**) as a baseline, a GFlowNet mixture with a perfect partition of 2^D parts, and GFlowNet mixture with a random partition of 2^D parts. We conduct each experiment using a random partition with 10 different seeds (which leads to 10 different partitions).

4.1.2 Results

(More experiments will be in the final paper, and the plots will be matplotlib plots that are easier to read)
The results of our experiments in the easy variant of a 32x32 grid (Figure 3) show that in a simple environment where a standard GFlowNet already performs well, a mixture of GFlowNet converges to a distribution that is almost as close to the target as a standard GFlowNet. Furthermore, results from the experiment in the hard variant of a 32x32 grid show that in environments where a standard GFlowNet struggles to find all the modes because the reward function is too sparse, using a mixture that isolate each mode into their own subset can help converge to more diverse solutions.

5 Molecule generation

5.1 Setup

We then evaluate the effectiveness of our method in a larger-scale task, fragment-based (Kumar et al. [2012]) molecule generation. As introduced in Bengio et al. [2021], the task is to generate molecules by linking molecule fragments to form a junction tree (Jin et al. [2020]). This is a challenging envrionment, with about 10^{16} states. The reward function is obtained via the pretrained proxy used in **?**, that predicts the binding energy of a molecule to the sEH protein. We extend an open source implementation of the environment[3]. This implementation is a slightly different from the one used in Bengio et al. [2021]. At each step, the agent has only 4 actions to choose from (compared to 100-2000 in the original implementation). However, put together these actions are approximately

[3]https://github.com/recursionpharma/gflownet

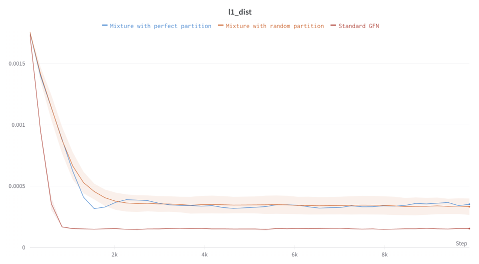

Figure 3: Evolution of L_1 distance between the learned distribution and the target distribution in the easy variant of a 32x32 grid. In blue, a mixture with a perfect partition, in orange, a mixture with a random partition, with minimum and maximum values obtained over 10 random seeds, and in red, a standard GFlowNet.

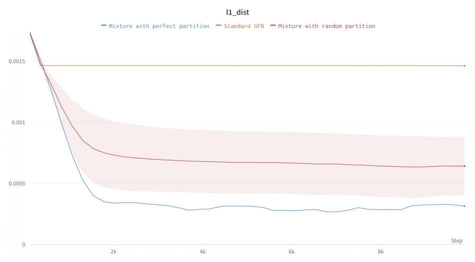

Figure 4: Evolution of L_1 distance between the learned distribution and the target distribution in the hard variant of a 32x32 grid. In blue, a mixture with a perfect partition, in red, a mixture with a random partition, with minimum and maximum values obtained over 10 random seeds, and in orange, a standard GFlowNet.

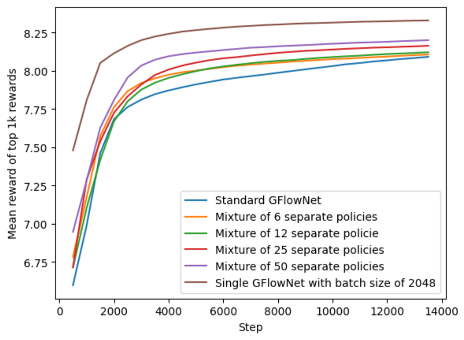

Figure 5: Evolution of the mean reward of the top 1000 molecules generated throughout training

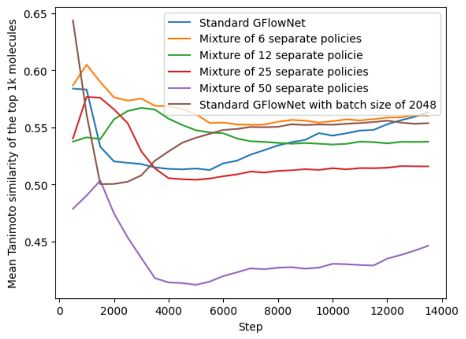

Figure 6: volution of the mean Tanimoto similarity of the top 1000 molecules generated throughout training

equivalent to the ones in Bengio et al. [2021]'s implementation, so we're able to generate more or less the same molecules, but with longer trajectories.

Experimental details We experimented with mixtures of 6, 12, 25 and 50 subordinate policies, each with a batch size of 64, and an off-policy ratio of 25% to help them converge. We also tested two baselines, a standard GFlowNet with a batch size of 64, and another standard GFlowNet with a batch size of 2048. Standard GFlowNets had an off-policy ratio of 0%. Policies were trained for 13500 steps.

5.2 Results

The results (Figures 5 and 6) show that compared to a standard GFlowNet with similar batch size, a mixture of GFlowNets generates better and more diverse molecules, and that this advantage scales with the amount of subpolicies. Also, we see that compared to a standard GFlowNet that is given about as much computing resources, a mixture generates top molecules that are more diverse, even though their average reward is a little lower.

References

E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. In *Neural Information Processing Systems (NeurIPS)*, 2021.

W. Jin, R. Barzilay, and T. Jaakkola. *Chapter 11. Junction Tree Variational Autoencoder for Molecular Graph Generation*, pages 228–249. 11 2020. ISBN 978-1-78801-547-9. doi: 10.1039/9781788016841-00228.

A. Kumar, A. Voet, and K. Zhang. Fragment based drug design: From experimental to computational approaches. *Current medicinal chemistry*, 19, 08 2012. doi: 10.2174/092986712803530467.

N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio. Trajectory balance: Improved credit assignment in gflownets. In *Neural Information Processing Systems (NeurIPS)*, 2022.