CFLOWNETS: CONTINUOUS CONTROL WITH GENERATIVE FLOW NETWORKS

Yinchuan Li[1], **Shuang Luo**[2], **Haozhi Wang**[3], **Jianye Hao**[1] *
[1]Huawei Noah's Ark Lab, Beijing, China
[2]Zhejiang University, Huangzhou, China
[3]Tianjin University, Tianjin, China
{liyinchuan, haojianye}@huawei.com

ABSTRACT

Generative flow networks (GFlowNets), as an emerging technique, can be used as an alternative to reinforcement learning for exploratory control tasks. GFlowNets aims to sample actions with a probability proportional to the reward, similar to sampling different candidates in an active learning fashion. However, existing GFlowNets cannot adapt to continuous control tasks because GFlowNets need to form a DAG and compute the flow matching loss by traversing the inflows and outflows of each node in the trajectory. In this paper, we propose generative continuous flow networks (CFlowNets) that can be applied to continuous control tasks. First, we present the theoretical formulation of CFlowNets. Then, a training framework for CFlowNets is proposed, including the action selection process, the flow approximation algorithm, and the continuous flow matching loss function. Afterward, we theoretically prove the error bound of the flow approximation. The error decreases rapidly as the number of flow samples increases. Finally, experimental results on continuous control tasks demonstrate the performance advantages of CFlowNets compared to many reinforcement learning methods, especially regarding exploration ability.

1 INTRODUCTION

As an emerging technology, generative flow networks (GFlowNets) (Bengio et al., 2021a;b) can make up for the shortcomings of reinforcement learning (Kaelbling et al., 1996; Sutton & Barto, 2018) on exploratory tasks. Specifically, based on the Bellman equation (Sutton & Barto, 2018), reinforcement learning is usually trained to maximize the expectation of future rewards; hence the learned policy is more inclined to sample action sequences with higher rewards. In contrast, the training goal of GFlowNets is to approximately sample candidate actions with probability proportional to a given reward function, rather than generating a single high-reward action sequence (Bengio et al., 2021a). This is more like sampling different candidates in an active learning setting (Bengio et al., 2021b), thus better suited for exploration tasks.

Unfortunately, GFlowNets currently cannot support continuous tasks. The main reason is because GFlowNets need to structure the state transitions of trajectories into a directed acyclic graph (DAG) structure. Each node in the graph structure corresponds to a different state, and actions correspond to transitions between different states, that is, an edge connecting different nodes in the graph. Obviously, the number of nodes in this graph structure is limited, and each edge can only correspond to one discrete action. This results in GFlowNets being limited to discrete and deterministic environments. However, in real environments, the state and action spaces are continuous for many tasks, such as quadrupedal locomotion (Kohl & Stone, 2004), autonomous driving (Kiran et al., 2021; Shalev-Shwartz et al., 2016; Pan et al., 2017), or dexterous in-hand manipulation (Andrychowicz et al., 2020). Moreover, the reward distributions corresponding to these environments may be mul-

*Corresponding Author: Shuang Luo
This work was completed while Shuang Luo and Haozhi Wang were members of the Huawei Noah's Ark Lab for advanced study.

timodal, requiring more diversity exploration. The needs of these environments closely match the strengths of GFlowNets, so addressing the discrete spatial constraints of GFlowNets is critical.

In this paper, we propose generative continuous flow networks, named CFlowNets for short, for continuous control tasks to generate policies that can be proportional to continuous reward functions. Applying GFlowNets to continuous control tasks is exceptionally challenging. In generative flow networks, the transition probability is defined as the ratio of action flow and state flow. For discrete state and action spaces, we can form a DAG and compute the state flow by traversing a node's incoming and outgoing flows. Conversely, it is impossible for continuous tasks to traverse all state-action pairs and corresponding rewards. To address this issue, we propose a method of sampling actions with probabilities roughly proportional to the output of the flow network. Then, we propose an efficient way to approximate the continuous inflows and outflows, where we use a deep neural network to predict the parent nodes of each state in the sampled trajectory. Finally, we present an approximate continuous flow matching loss function, which can efficiently train continuous flow networks. The main contributions of this paper are summarized as the following:

Main Contributions: 1) We present the theoritical formulation of CFlowNets and propose the continuous flow matching condition theorem. Based on this, a loss function for training CFlowNets is presented; 2) We present an efficient way to sample actions with probabilities approximately proportional to the output of the flow network, and propose a flow sampling approach to approximate continuous inflows and outflows, which allows us to construct a continuous flow matching loss; 3) We theoretically analyze the error bound between sampled flows and inflows/outflows, and the tail becomes minor as the number of flow samples increases; 4) We conduct experiments based on continuous control tasks to demonstrate that CFlowNets can outperform current state-of-the-art RL algorithms, especially in terms of exploration capabilities.

2 PRELIMINARIES

2.1 Markov Decision Process

A stochastic, discrete-time and sequential decision task can be described as a Markov Decision Process (MDP) , which is canonically formulated by the tuple:

$$M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle. \tag{1}$$

In the process, $s \in \mathcal{S}$ represents the state space of the environment. At each time step, agent receives a state s and selects an action a on the action space $\mathcal{A}$. This results in a transition to the next state s' according to the state transition function $P(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$. Then the agent gets the reward r based on the reward function $R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. A stochastic policy π maps each state to a distribution over actions $\pi(\cdot|s)$ and gives the probability $\pi(a|s)$ of choosing action a in state s. The agent interacts with the environment by executing the policy π and obtaining the admissible trajectories $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{n}$, where n is the trajectory length. The goal of an agent is to maximize the discounted return $\mathbb{E}_{s_{0:n}, a_{0:n}} [\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi]$, where $\mathbb{E}$ is the expectation over the distribution of the trajectories and $\gamma \in [0, 1)$ is the discount factor.

2.2 Generative Flow Network

GFlowNet sees the MDP as a flow network, that is, leverages the DAG structure of the MDP. Define $s' = T(s,a)$ and $F(s)$ as the node's transition and the total flow going through s. Define an edge/action flow $F(s,a) = F(s \to s')$ as the flow through an edge $s \to s'$. The training process of vanilla GFlowNets need to sum the flow of parents and children through nodes (states), which depends on the discrete state space and discrete action space. The framework is optimized by the following flow consistency equations:

$$\sum_{s,a:T(s,a)=s'} F(s,a) = R(s') + \sum_{a' \in \mathcal{A}(s')} F(s', a'), \tag{2}$$

which means that for any node s, the incoming flow equals the outgoing flow, which is the total flow $F(s)$ of node s.

3 CFLOWNETS: THEORETICAL FORMULATION

Considering a continuous task with tuple $(\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ denotes the continuous state space and $\mathcal{A}$ denotes the continuous action space. Define a trajectory $\tau = (s_1, ..., s_n)$ in this continuous task as a sequence sampled elements of $\mathcal{S}$ such that every transition $a_t : s_t \to s_{t+1} \in \mathcal{A}$. Further, we define an acyclic trajectory $\tau = (s_1, ..., s_n)$ as a trajectory satisfies the acyclic constraint: $\forall s_m \in \tau, s_k \in \tau, m \neq k$, we have $s_m \neq s_k$. Denote s_0 and s_f respectively as the initial state and the final state related with the continuous task $(\mathcal{S}, \mathcal{A})$, we define the complete trajectory as any sampled acyclic trajectory from $(\mathcal{S}, \mathcal{A})$ starting in s_0 and ending in s_f. Correspondingly, a transition $s \to s_f$ into the final state is defined as the terminating transition, and $F(s \to s_f)$ is a terminating flow.

A trajectory flow $F(\tau) : \tau \mapsto \mathbb{R}^+$ is defined as any nonnegative function defined on the set of complete trajectories τ. For each trajectory τ, the associated flow $F(\tau)$ contains the number of particles sharing the same path τ. In addition, the tuple $(\mathcal{S}, \mathcal{A}, F)$ is called a continuous flow network. Define the parent set $\mathcal{P}(s_t)$ of a state s_t as the set contains all of the direct parents of s_t that could make a direct transition to s_t under the acyclic constraint, i.e., $\mathcal{P}(s_t) = \{s \in \mathcal{S} : T(s, a \in \mathcal{A}) = s_t,\ T(s, a \in \mathcal{A}) \notin \{s_0, ..., s_{t-1}\}\}$, where $T(s, a) = s_t$ indicates an action a that could make a transition from state s to attain s_t, and $T(s, a \in \mathcal{A}) \notin \{s_0, ..., s_{t-1}\}$ means that the state cannot be transferred to a preexisting state, otherwise a cyclic will occur. Similarly, define the child set $\mathcal{C}(s_t)$ of a state s_t as the set contains all of the direct children of s_t that could make a direct transition from s_t under the acyclic constraint, i.e., $\mathcal{C}(s_t) = \{s \in \mathcal{S} : T(s_t, a \in \mathcal{A}) = s,\ T(s_t, a \in \mathcal{A}) \notin \{s_0, ..., s_t\}\}$. Then, we have the following continuous flow definitions.

Definition 1 (Continuous State Flow)**.** *The continuous state flow $F(s) : \mathcal{S} \mapsto \mathbb{R}$ is the integral of the complete trajectory flows passing through the state:*

$$F(s) = \int_{\tau : s \in \tau} F(\tau) \mathrm{d}\tau$$

Definition 2 (Continuous Inflows)**.** *For any state s_t, its inflows are the integral of flows that can reach state s_t under the acyclic constraint, i.e.,*

$$\int_{s \in \mathcal{P}(s_t)} F(s \to s_t) \mathrm{d}s = \int_{s, a : T(s,a) = s_t} F(s, a) \mathrm{d}s \mathrm{d}a,\ \textit{s.t.}\ T(s, a) \notin \{s_0, ..., s_{t-1}\}. \tag{3}$$

Definition 3 (Continuous Outflows)**.** *For any state s_t, the outflows are the integral of flows passing through state s_t with all possible actions $a \in \mathcal{A}$ under the acyclic constraint, i.e.,*

$$\int_{s \in \mathcal{C}(s_t)} F(s_t \to s) \mathrm{d}s = \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a,\ \textit{s.t.}\ T(s_t, a) \notin \{s_0, ..., s_t\}. \tag{4}$$

Based on the above definitions, we can define the transition probability $P(s \to s' | s)$ of edge $s \to s'$ as a special case of conditional probability introduced in Bengio et al. (2021b). In particular, the forward transition probability is given by

$$P_F(s_{t+1} | s_t) := P(s_t \to s_{t+1} | s_t) = \frac{F(s_t \to s_{t+1})}{F(s_t)}. \tag{5}$$

Similarly, the backwards transition probability is given by

$$P_B(s_t | s_{t+1}) := P(s_t \to s_{t+1} | s_{t+1}) = \frac{F(s_t \to s_{t+1})}{F(s_{t+1})}. \tag{6}$$

For any trajectory sampled from a continuous task $(\mathcal{S}, \mathcal{A})$, we have

$$\forall \tau = (s_1, ..., s_n), P_F(\tau) := \prod_{t=1}^{n-1} P_F(s_{t+1} | s_t) \tag{7}$$

$$\forall \tau = (s_1, ..., s_n), P_B(\tau) := \prod_{t=1}^{n-1} P_B(s_t | s_{t+1}), \tag{8}$$

and we further have

$$\forall s \in \mathcal{S}\backslash\{s_f\},\ \int_{s'\in\mathcal{C}(s)} P_F(s'|s)\mathrm{d}s' = 1 \tag{9}$$

$$\forall s \in \mathcal{S}\backslash\{s_0\},\ \int_{s'\in\mathcal{P}(s)} P_B(s'|s)\mathrm{d}s' = 1. \tag{10}$$

Given any trajectory $\tau = (s_0, ..., s_n, s)$ that starting in s_0 and ending in s, a Markovian flow (Bengio et al., 2021b) is defined as the flow that satisfies

$$P(s \to s'|\tau) = P(s \to s'|s) = P_F(s'|s),$$

and the corresponding flow network $(\mathcal{S}, \mathcal{A}, F)$ is called a Markovian flow network (Bengio et al., 2021b). Then, we present Theorem 1, which is proved in the appendix A.1.

Theorem 1 (Continuous Flow Matching Condition)**.** *Consider a non-negative function $\hat{F}(s, a)$ taking a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ as inputs. Then we have $\hat{F}$ corresponds to a flow if and only if the following continuous flow matching conditions are satisfied:*

$$\begin{aligned} \forall s' > s_0,\ \hat{F}(s') &= \int_{s\in\mathcal{P}(s')} \hat{F}(s \to s')\mathrm{d}s \\ \forall s' < s_f,\ \hat{F}(s') &= \int_{s''\in\mathcal{C}(s')} \hat{F}(s' \to s'')\mathrm{d}s''. \end{aligned} \tag{11}$$

Furthermore, $\hat{F}$ uniquely defines a Markovian flow F matching $\hat{F}$ such that

$$F(\tau) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{12}$$

Theorem 1 means that as long as any non-negative function satisfies the flow matching conditions, a unique flow is determined. Therefore, for sparse reward environments, i.e., $R(s) = 0,\ \forall s \neq s_f$, we can obtain the target flow by training a flow network that satisfies the flow matching conditions. Such learning machines are called generative Continuous Flow Networks, named CFlowNets for short, and we have the following continuous loss function:

$$\begin{aligned} \mathcal{L}(\tau) = \sum_{s_t=s_1}^{s_f} \Bigg(&\int_{s_{t-1}\in\mathcal{P}(s_t)} F(s_{t-1} \to s_t)\mathrm{d}s_{t-1} - \mathbb{I}_{s_t=s_f} R(s_t) \\ &- \mathbb{I}_{s_t\neq s_f} \int_{s_{t+1}\in\mathcal{C}(s_t)} F(s_t \to s_{t+1})\mathrm{d}s_{t+1} \Bigg)^2, \end{aligned} \tag{13}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function. However, obviously, the above continuous loss function cannot be directly applied in practice. Next, we propose a method to approximate the continuous loss function based on the sampled trajectories to obtain the flow model.

4 CFlowNets: Training Framework

For continuous tasks, it impossible to access all state-action pairs to calculate the continuous inflows and outflows. In the following, we propose the CFlowNets training framework to address this problem, which includes an action sampling process, a flow matching approximation process. Then, CFlowNets can be trained based on an approximate flow matching loss function.

4.1 Overall Framework

The overview framework of CFlowNets is shown in Figure 1, including the environment interaction, flow sampling, and training procedures. During the environment interaction phase (Left part of Figure 1), we sample an action probability buffer based on the forward-propagation of CFlowNets, from which we can sample an action with probability proportional to the reward. We name this

process the action selection procedure, as detailed in Section 4.2. After acquiring the action, the agent can interact with the environment to update the state, and this process repeats several steps until the complete trajectory is sampled. Once a buffer of complete trajectories is available, we randomly sample K actions and compute the child states to approximately calculate the outflows. For the inflows, we use these sampled actions together with the current state as the input to the deep neural network G to estimate the parent states. Based on these, we can approximately determine the inflows. We name this process the flow matching approximation procedure (Middle part of Figure 1), as detailed in Section 4.3. Finally, based on the approximate inflows and outflows, we can train a CFlowNet based on the continuous flow matching loss function (Right part of Figure 1), as details in Section 4.4. The pseudocode is provided in Appendix B.

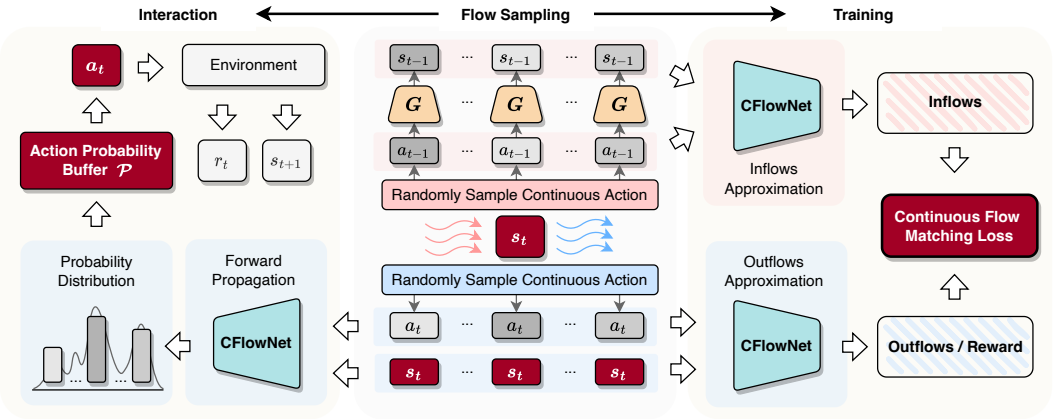


Figure 1: Overall framework of CFlowNets. **Left:** During the environment interaction phase, we sample actions to update states with probabilities proportional to the reward according to CFlowNet. **Middle:** We randomly sample actions to approximately calculate the inflows and outflows, where a DNN is used to estimate the parent states. **Right:** Continuous flow matching loss is used to train the CFlowNet based on making inflows equal to outflows or reward.

4.2 Action Selection Procedure

Starting from an empty set, CFlowNets aim to obtain complete trajectories $\tau = (s_0, s_1, ..., s_f) \in \mathcal{T}$ by iteratively sampling $a_t \sim \pi(a_t|s_t) = \frac{F(s_t,a_t)}{F(s_t)}$ with tuple $\{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{f}$. However, it is difficult to sample trajectories strictly according to the corresponding probability of a_t, since the actions are continuous, we cannot get the exact action probability distribution function based on the flow network $F(s_t, a_t)$. To solve this problem, at each state s_t, we first uniformly sample M actions from $\mathcal{A}$ and generate an action probability buffer $\mathcal{P} = \{F(s_t, a_i)\}_{i=1}^{M}$, which is used as an approximation of action probability distributions. Then we sample an action from $\mathcal{P}$ according to the corresponding probabilities of all actions. Obviously, actions with larger $F(s_t, a_i)$ will be sampled with higher probability. In this way, we approximately sample actions from a continuous distribution according to their corresponding probabilities.

Remark 1. *After the training process, for tasks that require a larger reward, we can sample actions with the maximum flow output in $\mathcal{P}$ during the test process to obtain a relatively higher reward. How the output of the flow model is used is flexible, and we can adjust it for different tasks.*

4.3 Flow Matching Approximation

Once a batch of trajectories $\mathcal{B}$ is available, to satisfy flow conditions, we require that for any node s_t, the inflows $\int_{s,a:T(s,a)=s_t} F(s,a)\mathrm{d}s\,\mathrm{d}a$ equals the outflows $\int_{a\in\mathcal{A}} F(s_t,a)\mathrm{d}a$, which is the total flow $F(s_t)$ of node s_t. However, obviously, we cannot directly calculate the continuous inflows and outflows to complete the flow matching condition. An intuitive idea is to discretize the inflows and outflows based on a reasonable approximation and match the discretized flows. To do this, we sample K actions independently and uniformly from the continuous action space $\mathcal{A}$ and calculate

corresponding $F(s_t, a_k), k = 1, ..., K$ as the outflows, i.e., we use the following approximation:

$$\int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \approx \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k), \tag{14}$$

where $\mu(\mathcal{A})$ denotes the measure of the continuous action space $\mathcal{A}$.

By contrast, an approximation of inflow is more difficult since we should find the parent state first. To solve this problem, we construct a transaction deep neural network G parameterized by ϕ with (s_{t+1}, a_t) as the input while s_t as the output, and train this network based on $\mathcal{B}$ with the MSE loss. The network G is usually easy to train, and we can obtain a high-precision network G through simple pre-training. As the training progresses, we can also occasionally update G based on the sampled trajectories to ensure accuracy. Then, the inflows can be calculated approximately:

$$\int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s \, \mathrm{d}a \approx \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_\phi(s_t, a_k), a_k). \tag{15}$$

Remark 2. *Recall that in Definitions 2 and 3, we define acyclic constraints that need to be satisfied when selecting possible actions and finding parents. But this constraint is only for the convenience of theoretical analysis. Here we ignore this constraint in equation 14 and equation 15 because the probability of picking continuous actions to form a cyclic is almost zero. This is understandable since* $\forall t,\ \mu(\{s_0, ..., s_t\}) = 0$ *and* $\mu(\mathcal{A}) = \mu(\mathcal{A} \backslash \{s_0, ..., s_t\})$.

Next, by assuming that the flow function $F(s, a)$ is Lipschitz continuous in Assumption 1, we could provide a non-asymptotic analysis for the error between the sample inflows/outflows and the true inflows/outflows. Theorem 2 establishes the error bound between the sample outflows (resp. inflows) and the actual outflows (resp. inflows) in the tail form and shows that the tail is decreasing exponentially. Furthermore, the tail gets much smaller with the increase of K, which means the sample outflows (resp. inflows) are a good estimation of the actual outflows (resp. inflows).

Assumption 1. *Assume the function* $F(s, a)$ *is Lipschitz continuous, i.e.,*

$$|F(s, a) - F(s, a')| \leq L_s ||a - a'||,\ a, a' \in \mathcal{A}, \tag{16}$$

where L_s *is a constant related to* s.

Theorem 2. *Let* $\{a_k\}_{k=1}^{K}$ *be sampled independently and uniformly from the continuous action space* $\mathcal{A}$. *Assume* $G_{\phi^\star}$ *can optimally output the actual state* s_t *with* (s_{t+1}, a_t). *For any bounded continuous action* $\mathcal{A}$ *and any state* $s_t \in \mathcal{S}$, *we have*

$$\mathbb{P}\left(\left| \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k) - \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \right| \geq t \right) \leq 2 \exp\left(-\frac{K t^2}{2(L_{s_t} \mu(\mathcal{A}) \mathrm{diam}(\mathcal{A}))^2} \right) \tag{17}$$

and

$$\mathbb{P}\left(\left| \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k) - \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s \, \mathrm{d}a \right| \geq t \right) \leq 2 \exp\left(-\frac{K t^2}{2(L_{s_t} \mu(\mathcal{A}) \mathrm{diam}(\mathcal{A}))^2} \right), \tag{18}$$

where L_{s_t} *is the Lipschitz constant of the function* $F(s_t, a)$ *and* $\mathrm{diam}(\mathcal{A})$ *denotes the diameter of the action space* $\mathcal{A}$.

4.4 Loss Function

Based on equation 14 and equation 15, the continuous loss function can be approximated by

$$\mathcal{L}_\theta(\tau) = \sum_{s_t = s_1}^{s_f} \left[\sum_{k=1}^{K} F_\theta(G_\phi(s_t, a_k), a_k) - \mathbb{I}_{s_t = s_f} \lambda R(s_t) - \mathbb{I}_{s_t \neq s_f} \sum_{k=1}^{K} F_\theta(s_t, a_k) \right]^2, \tag{19}$$

where θ is the parameter of the flow network $F(\cdot)$ and $\lambda = K/\mu(\mathcal{A})$. Note that the action set $\mathcal{A}$ changes slightly with the transition of nodes in the trajectory, because the action needs to satisfy the acyclic constraint $T(s_t, a) \in \{s_0, ..., s_t\}$. However, the measure of $\mathcal{A}$ is a constant because $\forall t,\ \mu(\mathcal{A}) = \mu(\mathcal{A} \backslash \{s_0, ..., s_t\})$. This shows that as long as the number of samples K is large enough, the discrete flow matching loss is equivalent to adding a fixed value shaping operation to the reward of the continuous flow matching loss, i.e., λ is more of a hyperparameter here. It doesn't need to be exactly equal to $K/\mu(\mathcal{A})$ but more like adding a fixed value shaping to the reward.

It is noteworthy that the magnitude of the state flow at different locations in the trajectory may not match. For example, the initial node flow is likely to be larger than the ending node flow. To solve this problem, inspired the log-scale loss introduced in GFlowNets (Bengio et al., 2021a), we can modified equation 19 into:

$$\mathcal{L}_\theta(\tau) = \sum_{s_t=s_1}^{s_f} \left\{ \log \left[\epsilon + \sum_{k=1}^{K} \exp F_\theta^{\log}(G_\phi(s_t, a_k), a_k) \right] - \log \left[\epsilon + \mathbb{I}_{s_t=s_f} \lambda R(s_t) + \mathbb{I}_{s_t \neq s_f} \sum_{k=1}^{K} \exp F_\theta^{\log}(s_t, a_k) \right] \right\}^2, \tag{20}$$

where ϵ is a hyper-parameter that helps to trade off small versus large flows and helps avoid the numerical problem of taking the logarithm of tiny flows.

5 Related Works

Generative Flow Networks. Generative flow networks are proposed to enhance exploration capabilities by generating policies that sample objects through discrete action sequences with probabilities proportional to a predefined reward function (Bengio et al., 2021b;a). Since the network only samples actions based on the distribution of the corresponding rewards, rather than focusing only on actions that maximize rewards such as reinforcement learning, it can perform well on tasks with more diverse reward distributions, and has been successfully applied to molecule generation (Bengio et al., 2021a; Malkin et al., 2022; Jain et al., 2022), discrete probabilistic modeling (Zhang et al., 2022b) and structure learning (Deleu et al., 2022). In Malkin et al. (2022), the trajectory balance loss is proposed for GFlowNets to explore the capabilities of previously used objectives. The connection between deep generative models and GFlowNets is discussed in Zhang et al. (2022a) through the lens of Markov trajectory learning. In Bengio et al. (2021b), an idea based on hybrid state is presented to make GFlowNets suitable for continuous tasks. This idea is mainly based on decomposing the continuous state space into a discrete state space plus continuous residuals. However, the decomposition process often requires gridding processing, which is not suitable for continuous spaces at high latitudes since the state and action dimensions would explode exponentially.

Continuous Reinforcement Learning. Policy gradient algorithms are widely used for reinforcement learning problems with continuous action spaces. The deterministic policy gradient (DPG) (Silver et al., 2014) algorithm is an actor-critic (Grondman et al., 2012; Rosenstein et al., 2004) method that uses an estimate of the learned value $Q(s, a)$ to train a deterministic policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$ parameterized by θ^μ. Compared with CFlowNets, the policy is updated by applying the chain rule to the expected return J from the start distribution with respect to the policy parameters:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{\mathcal{D}} \left[\nabla_{\theta^\mu} Q\left(s, a \mid \theta^Q\right)\big|_{a=\mu(s|\theta^\mu)} \right] \\ &= \mathbb{E}_{\mathcal{D}} \left[\nabla_a Q\left(s, a \mid \theta^Q\right)\big|_{a=\mu(s_t)} \nabla_{\theta^\mu} \mu\left(s \mid \theta^\mu\right) \right], \end{aligned} \tag{21}$$

where $\mathcal{D}$ is the replay buffer. The policy aims to maximize the expectation of future rewards, which are estimated by Q-learning. In this setting, the trajectories generated by the policy may be relatively homogeneous. However, for CFlowNets, by harnessing the power of the flow network, the probability of an action being sampled is proportional to the corresponding reward, resulting in more diverse trajectories that are beneficial for exploring the environment.

Later, deep DPG (DDPG) (Lillicrap et al., 2015) improves DPG and has good sample-efficient property but suffers from extreme brittleness and hyperparameter sensitivity. Therefore, it is difficult to extend DDPG to complex, high-dimensional tasks. To improve DDPG, twin delayed DDPG

(TD3) (Fujimoto et al., 2018) adopts an actor-critic framework and considers the interaction between value update and function approximation error and in the policy. There are also some policy gradient (Sutton et al., 1999; Kohl & Stone, 2004; Khadka & Tumer, 2018) based algorithms that can be adapted for continuous tasks, such as proximal policy optimization (PPO) (Schulman et al., 2017) algorithms, asynchronous advantage actor-critic (A3C) (Stooke & Abbeel, 2018), and importance weighted actor-learner architecture (IMPALA) (Espeholt et al., 2018). PPO has the benefits of trust region policy optimization (Schulman et al., 2015), enabling multiple batches of data to be updated together. Therefore, it is simpler to implement, more general, and has lower sample complexity. Recently, phasic policy gradient (PPG) (Cobbe et al., 2021) is proposed to decouple the training between policy and value function while keeping their feature sharing, and PPG optimizes each objective with an appropriate level of sample reuse to improve sample efficiency. All of these improved policy gradient methods can be classified as aiming at maximizing reward, so none of them are better suited for exploration tasks than CFlowNets.

Furthermore, some maximum entropy (Pitis et al., 2020; Haarnoja et al., 2018a; Hazan et al., 2019; Yarats et al., 2021) based reinforcement learning algorithms can also be adapted for continuous tasks, such as soft actor-critic (SAC) (Haarnoja et al., 2018b). By maximizing the expected reward and entropy, the actor network of SAC can successfully complete tasks while acting as randomly as possible. The difference between CFlowNets and SAC is: 1) Although the goal of SAC is to make policy proportional to return based on maximum entropy, the actor network of SAC outputs the mean and variance of the learned policy. Therefore, what SAC learns is not the true distribution of strictly proportional returns; 2) The return proportional to the policy in SAC can be understood as an accumulation of multiple samplings of the reward. This is different from being directly proportional to reward. In Bengio et al. (2021a), the difference between the two is discussed in Proposition 4, taking the tree structure as an example.

6 EXPERIMENTS

We investigate the performance of our CFlowNets by conducting experiments on several continuous control tasks with sparse rewards. The experimental setup is first introduced. Then we compare CFlowNets with a few state-of-the-art baseline RL algorithms, such as DDPG (Lillicrap et al., 2015), TD3 (Fujimoto et al., 2018), PPO (Schulman et al., 2017), and SAC (Haarnoja et al., 2018b).

6.1 EXPERIMENTAL SETUP

We conduct our experiments on three continuous control tasks: Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse. Point-Robot-Sparse is a continuous navigation task. FigureThe visualization of these environments is shown in Figure 4. In this task, the agent starts at the starting coordinate $(0, 0)$ and moves towards the target coordinate one step at a time. This experiment has two target coordinates $(5, 10)$ and $(10, 5)$. The environment has a maximum episode length of 12, and the environment returns a reward only when the last step is reached. Rewards are issued by measuring the distance between the agent's current position and the target node, and the closer the distance, the greater the reward. Each time the agent can choose to take a step from any angle to the upper right.

Both Reacher-Goal-Sparse and Swimmer-Sparse are adapted from OpenAI Gym's MuJoCo environment. In the Reacher-Goal-Sparse environment, "Reacher" is a two-jointed robotic arm. The goal is to move the robot's end effector (called fingertip) close to a randomly generated target. The "swimmer" is suspended in a two-dimensional pool, and the goal is to move as fast as possible towards the right or left by taking the action that applies torque between links of the rotors. We set the maximum number of steps to 50 for these two environments. For Reacher-Goal-Sparse, when the last step is reached, the environment returns a reward that measures how far the agent is from the randomly generated target. The closer the agent is to the target, the greater the reward. For Swimmer-Sparse, the farther to the left or right from the starting point, the greater the reward returned. More details of parameter setting are provide in Appendix C.

6.2 EXPERIMENTAL PERFORMANCE

Figure 2 illustrates the distributions of learned policies for CFlowNets and RL algorithms. All curves are max-min normalized. The gray curve is the ground truth of reward distribution generated by the agent's different actions when it goes to coordinates $(7, 7)$, which indicates that the optimal action here is to go right or up. The red curve shows the flow network output of CFlowNets under different actions, indicating that CFlowNets have an excellent fitting ability to the reward. In contrast, other reinforcement learning algorithms have difficulty fitting the actual reward distribution well.

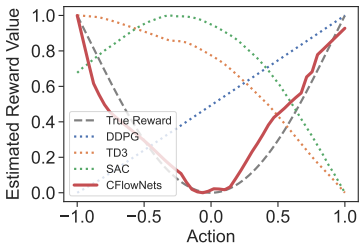


Figure 2: Reward distributions on Point-Robot-Sparse Task.

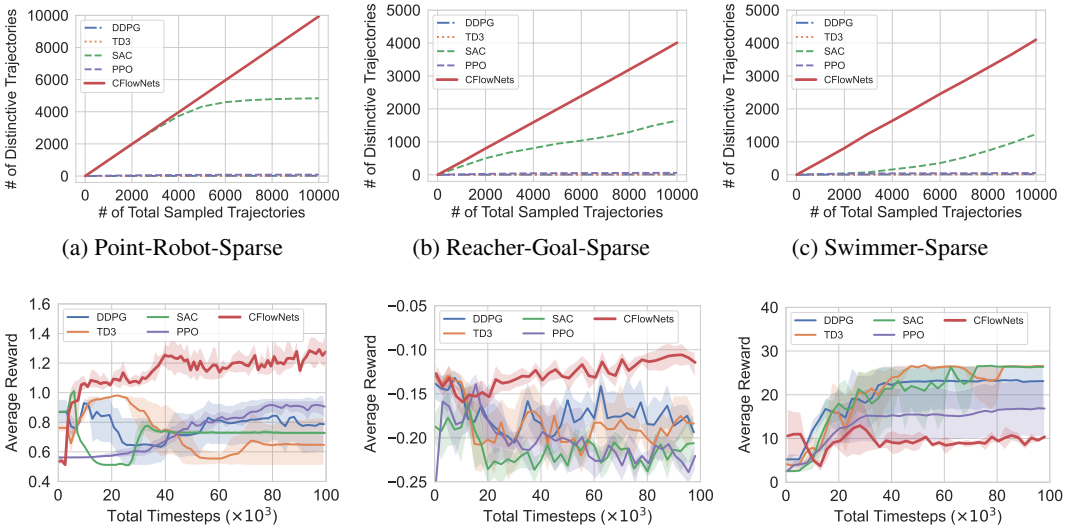
Figures 3(a)-(c) show the number of valid-distinctive trajectories explored as training progresses in Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse environment, respectively. After a certain number of training epochs, 10000 trajectories are collected. A valid-distinctive trajectory is defined as a reward above a threshold δ_r while the MSE between the trajectory and other trajectories is greater than another threshold δ_{mse}. That is, if the returns of both trajectories are high, but the two are close and the MSE is small, we consider it only one valid-distinctive exploration. δ_r in Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse is set as 0.5, -0.2, 5.0, respectively. δ_{mse} in Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse is set as 0.02, 4.0, 1.0, respectively. As can be seen from the figure, DDPG, TD3 and PPO have the worst exploration ability, only one valid-distinctive trajectory is generated. SAC explores better at the beginning of training, and decreases as the training progresses and gradually converges. In contrast, the exploration ability of CFlowNets is very outstanding, the number of trajectories explored far exceeds other algorithms, and the exploration ability has been stable as the training progresses.

Figures 3(d)-(f) indicate the rewards during the training process in Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse environment, respectively. The shaded region represents 95% confidence interval across 5 runs. Figure 3(d) and Figure 3(e) show that CFlowNets has the fastest and more stable upward trend, and the final reward is ahead of that of other algorithms by a large margin. In contrast, CFlowNets do not perform as well as other algorithms in Figure 3(f). Since the rewards in Point-Robot-Sparse and Reacher-Goal-Sparse are more evenly distributed, so these two tasks are more inclined to exploration. CFlowNets has better exploration ability and hence can converge stably. As for Swimmer-Sparse, its reward distribution is relatively steep, and sampling near the maximum reward can achieve faster convergence. It is reasonable for CFlowNets to perform worse than RL on this task in terms of reward. However, in this environment, CFN can still maintain a good exploration ability.

7 Conclusion

In this paper, we propose generative continuous flow networks to enhance exploration in continuous control tasks. The theoretical formulation of CFlowNets is first presented. Then, a training framework for CFlowNets is proposed, including the action selection process, the flow approximation algorithm, and the continuous flow matching loss function. Theoretical analysis shows that the error of the flow approximation decreases rapidly as the number of flow samples increases. Experimental results on continuous control tasks illustrate the performance advantages of CFlowNets compared to many reinforcement learning methods. Especially in the exploration ability, the effect of CFlowNets far exceeds other state-of-the-art reinforcement learning algorithms.

Limitations: Similar to GFlowNets, CFlowNets aims to sample actions with probabilities proportional to the reward distribution, rather than selecting actions with maximizing rewards. Therefore, CFlowNets are more suitable for exploration-biased tasks. It does not perform as well as reinforcement learning on tasks that aim to maximize reward. Of course, the purpose of CFlowNets is not to completely replace reinforcement learning, but as a supplement to reinforcement learning, giving a new option for continuous control tasks.



(a) Point-Robot-Sparse (b) Reacher-Goal-Sparse (c) Swimmer-Sparse

(d) Point-Robot-Sparse (e) Reacher-Goal-Sparse (f) Swimmer-Sparse

Figure 3: Comparison results of CFlowNets, DDPG, TD3, SAC and PPO on Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse tasks. **Top:** Number of valid-distinctive trajectories generated under 10000 explorations. **Bottom:** The average reward of different methods.

REFERENCES

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation, 2021a.

Yoshua Bengio, Tristan Deleu, Edward J. Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations, 2021b.

Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pp. 2020–2027. PMLR, 2021.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. *arXiv preprint arXiv:2202.13903*, 2022.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 1851–1860. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018b.

Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pp. 2681–2691. PMLR, 2019.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pp. 2619–2624. IEEE, 2004.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *arXiv preprint arXiv:2201.13259*, 2022.

Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pp. 7750–7761. PMLR, 2020.

Michael T Rosenstein, Andrew G Barto, Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch. Supervised actor-critic reinforcement learning. *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, pp. 359–380, 2004.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. PMLR, 2014.

Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pp. 11920–11931. PMLR, 2021.

Dinghuai Zhang, Ricky TQ Chen, Nikolay Malkin, and Yoshua Bengio. Unifying generative models with gflownets. *arXiv preprint arXiv:2209.02606*, 2022a.

Dinghuai Zhang, Nikolay Malkin, Zhen Liu, Alexandra Volokhova, Aaron Courville, and Yoshua Bengio. Generative flow networks for discrete probabilistic modeling. *arXiv preprint arXiv:2202.01361*, 2022b.

A PROOFS

A.1 PROOF OF THEOREM 1

Theorem 1. (Continuous Flow Matching Condition). *Consider a non-negative function $\hat{F}(s, a)$ taking a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ as inputs. Then we have $\hat{F}$ corresponds to a flow if and only if the following continuous flow matching conditions are satisfied:*

$$\begin{aligned} \forall s' > s_0,\ \hat{F}(s') &= \int_{s \in \mathcal{P}(s')} \hat{F}(s \to s') \mathrm{d}s \\ \forall s' < s_f,\ \hat{F}(s') &= \int_{s'' \in \mathcal{C}(s')} \hat{F}(s' \to s'') \mathrm{d}s''. \end{aligned} \tag{22}$$

Furthermore, $\hat{F}$ uniquely defines a Markovian flow F matching $\hat{F}$ such that

$$F(\tau) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{23}$$

Proof. The proof is an extension of that of Proposition 19 in Bengio et al. (2021b) to the continuous case. We first prove the necessity part of the proof. Given a flow network, for non-initial and non-final nodes on a trajectory, the set of complete trajectories passing through state s' is the union of the sets of trajectories going through $s \to s'$ for all $s \in \mathcal{P}(s')$, and also is the union of the sets of trajectories going through $s' \to s''$ for all $s'' \in \mathcal{C}(s')$, i.e.,

$$\{\tau \in \mathcal{T} : s' \in \tau\} = \bigcup_{s \in \mathcal{P}(s')} \{\tau \in \mathcal{T} : s \to s' \in \tau\} = \bigcup_{s'' \in \mathcal{C}(s')} \{\tau \in \mathcal{T} : s' \to s'' \in \tau\}.$$

Then we have

$$F(s') = \int_{\tau : s' \in \tau} F(\tau) \mathrm{d}\tau = \int_{s \in \mathcal{P}(s')} \int_{\tau : s \to s' \in \tau} F(\tau) \mathrm{d}\tau \mathrm{d}s = \int_{s \in \mathcal{P}(s')} F(s \to s') \mathrm{d}s,$$

and

$$F(s') = \int_{\tau : s' \in \tau} F(\tau) \mathrm{d}\tau = \int_{s'' \in \mathcal{C}(s')} \int_{\tau : s' \to s'' \in \tau} F(\tau) \mathrm{d}\tau \mathrm{d}s'' = \int_{s'' \in \mathcal{C}(s')} F(s' \to s'') \mathrm{d}s''.$$

Then we finish the necessity part. Next we show sufficiency. Let $\hat{Z} = \hat{F}(s_0)$ be the partition function and $\hat{P}_F$ be the forward probability function, then there exists a unique Markovian flow F with forward transition probability function $P_F = \hat{P}_F$ and partition function Z according to Proposition 18 in Bengio et al. (2021b), and such that

$$F(\tau) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t | s_{t-1}) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}, \tag{24}$$

where $s_{n+1} = s_f$. In addition, according to Lemma 1, we have

$$\int_{\tau \in \mathcal{T}_{0,s}} P_B(\tau) \mathrm{d}\tau = \int_{\tau \in \mathcal{T}_{0,s}} \prod_{s_t \to s_{t+1} \in \tau} P_B(s_t | s_{t+1}) \mathrm{d}\tau = 1.$$

Lemma 1. *Considering a continuous task $(\mathcal{S}, \mathcal{A})$, where we have the transition probabilities defined in equation 5 and equation 6. Define $\mathcal{T}_{s,f}$ and $\mathcal{T}_{0,s}$ as the set of trajectories sampled from a continuous task starting in s and ending in s_f; and starting in s_0 and ending in s, respectively. Then we have*

$$\forall s \in \mathcal{S} \backslash \{s_f\},\ \int_{\tau \in \mathcal{T}_{s,f}} P_F(\tau) \mathrm{d}\tau = 1 \tag{25}$$

$$\forall s \in \mathcal{S} \backslash \{s_0\},\ \int_{\tau \in \mathcal{T}_{0,s}} P_B(\tau) \mathrm{d}\tau = 1. \tag{26}$$

Thus, we have for $s' \neq s_0$:

$$F(s') = \hat{Z} \int_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_F(s_{t+1}|s_t) \mathrm{d}\tau$$
$$= \hat{Z} \frac{\hat{F}(s')}{\hat{F}(s_0)} \int_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_B(s_t|s_{t+1}) \mathrm{d}\tau = \hat{F}(s'). \tag{27}$$

Combine equation 27 with $P_F = \hat{P}_F$ yields $\forall s \to s' \in \mathcal{A}, F(s \to s')$. Finally, according to Proposition 16 in Bengio et al. (2021b), for any Markovian flow F' matching $\hat{F}$ on states and edges, we have $F'(\tau) = F(\tau)$, which shows the uniqueness property. Then we complete the proof. □

A.2 Proof of Theorem 2

Theorem 2. *Let $\{a_k\}_{k=1}^K$ be sampled independently and uniformly from the continuous action space $\mathcal{A}$. Assume $G_{\phi^\star}$ can optimally output the actual state s_t with (s_{t+1}, a_t). For any bounded continuous action $\mathcal{A}$ and any state $s_t \in \mathcal{S}$, we have*

$$\mathbb{P}\left(\left| \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k) - \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \right| \geq t \right) \leq 2 \exp\left(-\frac{Kt^2}{2(L_{s_t} \mu(\mathcal{A}) \mathrm{diam}(\mathcal{A}))^2} \right) \tag{28}$$

and

$$\mathbb{P}\left(\left| \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k) - \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s\, \mathrm{d}a \right| \geq t \right)$$
$$\leq 2 \exp\left(-\frac{Kt^2}{2(L_{s_t} \mu(\mathcal{A}) \mathrm{diam}(\mathcal{A}))^2} \right), \tag{29}$$

where L_{s_t} is the Lipschitz constant of the function $F(s_t, a)$ and $\mathrm{diam}(\mathcal{A})$ denotes the diameter of the action space $\mathcal{A}$.

Proof. First, we show that the expectation of sample outflow is the true outflow and the expectation of sample inflow is the true inflow in Lemma 2.

Lemma 2. *Let $\{a_k\}_{k=1}^K$ be sampled independently and uniformly from the continuous action space $\mathcal{A}$. Assume $G_{\phi^\star}$ can optimally output the actual state s_t with (s_{t+1}, a_t). Then for any state $s_t \in \mathcal{S}$, we have*

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k) \right] = \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \tag{30}$$

and

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k) \right] = \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s\, \mathrm{d}a. \tag{31}$$

Then, define the following terms:

$$\Gamma_k = \frac{\mu(\mathcal{A})}{K} F(s_t, a_k) - \frac{1}{K} \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a = \frac{1}{K} \int_{a \in \mathcal{A}} \left[F(s_t, a_k) - F(s_t, a) \right] \mathrm{d}a \tag{32}$$

and

$$\Lambda_k = \frac{\mu(\mathcal{A})}{K} F(G_{\phi^\star}(s_t, a_k), a_k) - \frac{1}{K} \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s\, \mathrm{d}a \tag{33}$$
$$= \frac{1}{K} \int_{s,a:T(s,a)=s_t} \left[F(G_{\phi^\star}(s_t, a_k), a_k) - F(s, a) \right] \mathrm{d}s\, \mathrm{d}a \tag{34}$$

Note that the variables $\{\Gamma_k\}_{k=1}^K$ are independent and $\mathbb{E}[\Gamma_k] = 0, k = 1, \ldots, K$ according to Lemma 2. So the following equations hold

$$\mathbb{P}\left(\Big|\frac{\mu(\mathcal{A})}{K}\sum_{k=1}^{K} F(s_t, a_k) - \int_{a\in\mathcal{A}} F(s_t, a)\mathrm{d}a\Big| \geq t\right) = \mathbb{P}\left(\Big|\sum_{k=1}^{K}\Gamma_k\Big| \geq t\right) \tag{35}$$

and

$$\mathbb{P}\left(\Big|\frac{\mu(\mathcal{A})}{K}\sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k) - \int_{s,a:T(s,a)=s_t} F(s, a)\mathrm{d}s\,\mathrm{d}a\Big| \geq t\right) = \mathbb{P}\left(\Big|\sum_{k=1}^{K}\Lambda_k\Big| \geq t\right). \tag{36}$$

Since $F(s, a)$ is a Lipschitz function, we have

$$|\Gamma_k| \leq \frac{1}{K}\int_{a\in\mathcal{A}} \big|F(s_t, a_k) - F(s_t, a)\big|\mathrm{d}a \leq \frac{L_{s_t}}{K}\int_{a\in\mathcal{A}} ||a_k - a||\mathrm{d}a \leq \frac{L_{s_t}\mu(\mathcal{A})\mathrm{diam}(\mathcal{A})}{K}. \tag{37}$$

Together with the fact that for any pair of (s,a) satisfying $T(s, a) = s_t$, s is unique if we fix a, we have

$$|\Lambda_k| \leq \frac{1}{K}\int_{s,a:T(s,a)=s_t} \big|F(G_{\phi^\star}(s_t, a_k), a_k) - F(s, a)\big|\mathrm{d}s\,\mathrm{d}a \tag{38}$$

$$\leq \frac{L_{s_t}}{K}\int_{s,a:T(s,a)=s_t} ||a_k - a||\mathrm{d}s\,\mathrm{d}a \leq \frac{L_{s_t}\mu(\mathcal{A})\mathrm{diam}(\mathcal{A})}{K}. \tag{39}$$

Lemma 3 (Hoeffding's inequality, Vershynin (2018))**.** *Let $x_1, \ldots, x_K$ be independent random variables. Assume the variables $\{x_k\}_{k=1}^K$ are bounded in the interval $[T_l, T_r]$. Then for any $t > 0$,we have*

$$\mathbb{P}\left(\Big|\sum_{k=1}^{K}(x_k - \mathbb{E}x_k)\Big| \geq t\right) \leq 2\exp\left(-\frac{2t^2}{K(T_r - T_l)^2}\right). \tag{40}$$

Incorporating $T_r = \frac{L_{s_t}\mu(\mathcal{A})\mathrm{diam}(\mathcal{A})}{K}$ and $T_r = -\frac{L_{s_t}\mu(\mathcal{A})\mathrm{diam}(\mathcal{A})}{K}$ in Lemma 3 completes the proof. □

A.3 Proof of Lemma 1

Lemma 1. *Considering a continuous task $(\mathcal{S}, \mathcal{A})$, where we have the transition probabilities defined in equation 5 and equation 6. Define $\mathcal{T}_{s,f}$ and $\mathcal{T}_{0,s}$ as the set of trajectories sampled from a continuous task starting in s and ending in s_f; and starting in s_0 and ending in s, respectively. Then we have*

$$\forall s \in \mathcal{S}\backslash\{s_f\},\ \int_{\tau\in\mathcal{T}_{s,f}} P_F(\tau)\mathrm{d}\tau = 1 \tag{41}$$

$$\forall s \in \mathcal{S}\backslash\{s_0\},\ \int_{\tau\in\mathcal{T}_{0,s}} P_B(\tau)\mathrm{d}\tau = 1. \tag{42}$$

Proof. We show by strong induction that equation 41 holds, mainly following the proof of Lemma 5 in Bengio et al. (2021b), and then extending to equation 42 is trivial. Define d as the maximum trajectory length in $\mathcal{T}_{s,f}, s \neq s_f$, we have:

Base cases: If $d = 1$, then

$$\int_{\tau\in\mathcal{T}_{s,f}} P_F(\tau)\mathrm{d}\tau = P_F(s \to s_f) = 1$$

holds by noting $\mathcal{T}_{s,f} = \{(s \to s_f)\}$.

Induction steps: Consider $d > 1$, by noting equation 9 we have

$$\int_{\tau \in \mathcal{T}_{s,f}} P_F(\tau) \mathrm{d}\tau = \int_{s' \in \mathcal{C}(s)} \int_{\tau \in \mathcal{T}_{s \to s',f}} P_F(\tau) \mathrm{d}\tau \mathrm{d}s' \tag{43}$$

$$= \int_{s' \in \mathcal{C}(s)} \int_{\tau \in \mathcal{T}_{s',f}} P_F(s'|s) P_F(\tau) \mathrm{d}\tau \mathrm{d}s' \tag{44}$$

$$= \int_{s' \in \mathcal{C}(s)} P_F(s'|s) \mathrm{d}s' \int_{\tau \in \mathcal{T}_{s',f}} P_F(\tau) \mathrm{d}\tau = 1, \tag{45}$$

where the last equality follows by the induction hypotheses. □

A.4 Proof of Lemma 2

Lemma 2. *Let $\{a_k\}_{k=1}^K$ be sampled independently and uniformly from the continuous action space $\mathcal{A}$. Assume $G_{\phi^\star}$ can optimally output the actual state s_t with (s_{t+1}, a_t). Then for any state $s_t \in \mathcal{S}$, we have*

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k)\right] = \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \tag{46}$$

and

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k)\right] = \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s \, \mathrm{d}a. \tag{47}$$

Proof. Since $\{a_k\}_{k=1}^K$ is sampled independently and uniformly from the continuous action space $\mathcal{A}$, then we have

$$\mathbb{E}\left[F(s_t, a_k)\right] = \frac{1}{\mu(\mathcal{A})} \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a. \tag{48}$$

Therefore, we obtain

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(s_t, a_k)\right] = \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} \mathbb{E}\left[F(s_t, a_k)\right] \tag{49}$$

$$= \int_{a \in \mathcal{A}} F(s_t, a) \mathrm{d}a \tag{50}$$

Note that for any pair of (s,a) satisfying $T(s, a) = s_t$, s is unique if we fix a. So we have

$$\mathbb{E}\left[F(G_{\phi^\star}(s_t, a_k), a_k)\right] = \frac{1}{\mu(\mathcal{A})} \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s \, \mathrm{d}a. \tag{51}$$

Therefore, we get

$$\mathbb{E}\left[\frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} F(G_{\phi^\star}(s_t, a_k), a_k)\right] = \frac{\mu(\mathcal{A})}{K} \sum_{k=1}^{K} \mathbb{E}\left[F(G_{\phi^\star}(s_t, a_k), a_k)\right]$$

$$= \int_{s,a:T(s,a)=s_t} F(s, a) \mathrm{d}s \, \mathrm{d}a.$$

□

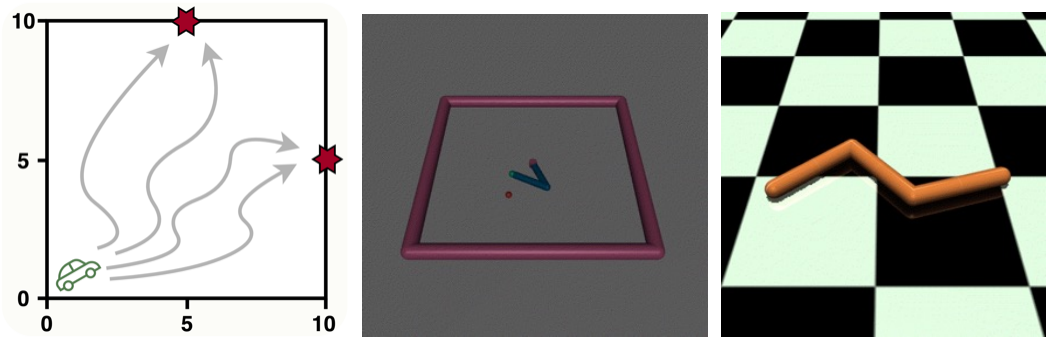
B Pseudocode of CFlowNets

For clarity, we show pseudocode for CFlowNets in Algorithm 1.

Algorithm 1 Generative Continuous Flow Networks (CFlowNets) Algorithm

Initialize: Flow network θ; a pretrained transaction network ϕ; and empty buffer $\mathcal{D}$ and $\mathcal{P}$

repeat
  Set $t = 0$, $s = s_0$
  while $s \neq terminal$ and $t<T$ **do**
    Uniformly sample M actions $\{a_i\}_{i=1}^{M}$ from action space $\mathcal{A}$
    Compute edge flow $F_\theta(s_t, a_i)$ for each $a_i \in \{a_i\}_{i=1}^{M}$ to generate $\mathcal{P}$
    Sample $a_t \sim \mathcal{P}$ and execute a_t in the environment to obtain r_{t+1} and s_{t+1}
    $t = t + 1$
  end while
  Store episodes $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{T}$ in replay buffer $\mathcal{D}$
  [Optional] Fine-tuning transaction network ϕ based on $\mathcal{D}$
  Sample a random minibatch $\mathcal{B}$ of episodes from $\mathcal{D}$
  Uniformly sample K actions $\{a_k\}_{k=1}^{K}$ from action space $\mathcal{A}$ for each state in $\mathcal{B}$
  Compute parent states according to $\{G_\phi(s, a_k)\}_{k=1}^{K}$ for each state in $\mathcal{B}$
  Inflows:
    $\log[\epsilon + \sum_{k=1}^{K} \exp F_\theta^{\log}(G_\phi(s_t, a_k), a_k)]$
  Outflows or reward:
    $\log[\epsilon + \mathbb{I}_{s_t=s_f} \lambda R(s_t) + \mathbb{I}_{s_t \neq s_f} \sum_{k=1}^{K} \exp F_\theta^{\log}(s_t, a_k)]$
  Update flow network F_θ according to equation 20
until convergence



(a) Point-Robot-Sparse (b) Reacher-Goal-Sparse (c) Swimmer-Sparse

Figure 4: Visualization of Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse tasks.

C Experimental setup

C.1 Visualization of environment

As shown in Figure 4, we provide the visualization of Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse tasks.

C.2 Baselines

We compare our proposed CFlowNets to the following baselines:

- Deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015). `https://github.com/sfujim/TD3/blob/master/DDPG.py`
- Twin delayed deep deterministic policy gradient (TD3) (Fujimoto et al., 2018). `https://github.com/sfujim/TD3`
- Soft actor-critic (SAC) (Haarnoja et al., 2018b). `https://github.com/denisyarats/pytorch_sac/`

- Proximal policy optimization (PPO) (Schulman et al., 2017). https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/ppo/ppo.py

C.3 HYPER-PARAMETER

We provide the hyper-parameter of all compared methods under different environments in Table 1, Table 2, Table 3, Table 4, and Table 5.

Table 1: Hyper-parameter of CFlowNets under different environments

	Point-Robot-Sparse	Reacher-Goal-Sparse	Swimmer-Sparse
Total Timesteps	100,000	100,000	100,000
Start Traning Timestep	4,000	7,500	7,500
Max Episode Length	12	50	50
Flow Network Hidden Layers	[256,256]	[256,256]	[256,256]
Transaction Network Hidden Layers	[256,256,256]	[256,256,256]	[256,256,256]
Optimizer	Adam	Adam	Adam
Learning Rate	0.0003	0.0003	0.0003
Batchsize	128	128	128
Number of Sample Flows	100	100	100
Action Probability Buffer Size	1,000	1,0000	10,000
Replay Buffer Size	8,000	2,000	2,000
ϵ	1.0	1.0	1.0

Table 2: Hyper-parameter of DDPG under different environments

	Point-Robot-Sparse	Reacher-Goal-Sparse	Swimmer-Sparse
Total Timesteps	100,000	100,000	100,000
Start Traning Timestep	4,000	7,500	7,500
Max Episode Length	12	50	50
Actor Network Hidden Layers	[256,256]	[256,256]	[256,256]
Critic Network Hidden Layers	[256,256]	[256,256]	[256,256]
Optimizer	Adam	Adam	Adam
Learning Rate	0.0003	0.0003	0.0003
Batchsize	256	256	256
Discount Factor	0.99	0.99	0.99
Replay Buffer Size	100,000	100,000	100,000
Target Network Update Rate	0.005	0.005	0.005

Table 3: Hyper-parameter of TD3 under different environments

	Point-Robot-Sparse	Reacher-Goal-Sparse	Swimmer-Sparse
Total Timesteps	100,000	100,000	100,000
Start Traning Timestep	4,000	7,500	7,500
Max Episode Length	12	50	50
Actor Network Hidden Layers	[256,256]	[256,256]	[256,256]
Critic Network Hidden Layers	[256,256]	[256,256]	[256,256]
Optimizer	Adam	Adam	Adam
Learning Rate	0.0003	0.0003	0.0003
Batchsize	128	128	128
Discount Factor	0.99	0.99	0.99
Replay Buffer Size	100,000	100,000	100,000
Gaussian Exploration Noise	0.1	0.1	0.1
Target Network Update Rate	0.005	0.005	0.005

Table 4: Hyper-parameter of SAC under different environments

	Point-Robot-Sparse	Reacher-Goal-Sparse	Swimmer-Sparse
Total Timesteps	100,000	100,000	100,000
Start Traning Timestep	4,000	7,500	7,500
Max Episode Length	12	50	50
Actor Network Hidden Layers	[256,256]	[256,256]	[256,256]
Critic Network Hidden Layers	[256,256]	[256,256]	[256,256]
Optimizer	Adam	Adam	Adam
Learning Rate	0.0003	0.0003	0.0003
Batchsize	1024	1024	1024
Discount Factor	0.99	0.99	0.99
Replay Buffer Size	100,000	100,000	100,000
Target Update Interval	1	1	1

Table 5: Hyper-parameter of PPO under different environments

	Point-Robot-Sparse	Reacher-Goal-Sparse	Swimmer-Sparse
Total Timesteps	100,000	100,000	100,000
Max Episode Length	12	50	50
Policy Network Hidden Layers	[64,64]	[64,64]	[64,64]
Value Network Hidden Layers	[64,64]	[64,64]	[64,64]
Optimizer	Adam	Adam	Adam
Learning Rate	0.0003	0.0003	0.0003
Batchsize	64	64	64
Discount Factor	0.99	0.99	0.99
GAE Parameter	0.95	0.95	0.95
Timesteps per Update	2048	2048	2048
Number of Epochs	10	10	10
Clipping Parameter	0.2	0.2	0.2
Value Loss Coefficient	0.5	0.5	0.5