Generating Random Variables and Stochastic Processes

Generative Flow Networks (GFlowNets)

a.b@mila.quebec

Pascal[a] Jr TIKENG[b] N.

Mila, DIRO, UdeM

August 15, 2023

Outline

- (*) : prerequisites
- The goal here is to present (only) GflowNets, but I think it is necessary to understand also (*) : I usually call GflowNets *"parameterized MCMC"*, let try to see why.

Notations

- RV, r.v. : Random variable (X, U)
- pdf : probability density function (f, g, π)
- cdf : cumulative distribution function (F, G)
- RNG : Random Number Genarator
- MCMC : Markov Chain Monte Carlo
- MH : Metropolis-Hasting
- GFlowNets : Generative Flow Networks

I - Problem to solve: Sampling from a probability distribution

Let π a probability distribution defined on a space $\mathcal{X}$ (let's ignore the notion of probability space for the moment).

- We would like to have an algorithm that generates elements from $\mathcal{X}$ such that the probability of an $x \in \mathcal{X}$ being generated is equal to $\pi(x)$
- In other words, if the algorithm generates a list of N elements, we would like to have a number $N\pi(x)$ of each $x \in \mathcal{X}$ in the list, ie a proportion $\pi(x)$ of each x in the list
- This is called sampling from a probability distribution and it is different from solving directly (exploration vs exploitation) $\arg\max_{x \in \mathcal{X}} \pi(x)$
- The high probability states will therefore be more generated than low probability states.
- In some cases, π is not explicitly known, but $R : \mathcal{X} \mapsto \mathbb{R}_+$ such that

$$\pi(x) \propto R(x) \ \forall x \in \mathcal{X}$$

I - Problem to solve : Example

Generate (randomly) **positive** multiples $x_1, \ldots, x_m$ of a positive integer n, with each x_i being of length less than or equal to $\ell > 2$.

Attempt 1

- State space

$$\mathcal{X} = \{nk, k \in \mathbb{N}, nk \leq 10^\ell - 1\}$$

- Reward function (Uniform over $\mathcal{X}$)

$$R = \mathbb{1}_{\mathcal{X}}$$

- Canonical partition function

$$Z = \sum_{x \in \mathcal{X}} R(x) = |\mathcal{X}|$$

- Probability distribution (Uniform over $\mathcal{X}$)

$$\pi(x) = \frac{R(x)}{Z} = \frac{1}{|\mathcal{X}|} \mathbb{1}_{\mathcal{X}}(x)$$

I - Problem to solve: Example

Generate (randomly) **positive** multiples $x_1, \dots, x_m$ of a positive integer n, with each x_i being of length less than or equal to $\ell > 2$.

Attempt 2

- State space
$$\mathcal{X} = \{0, ..., 10^{\ell} - 1\}$$
- Reward function
$$R(x) = \lambda^{-\beta \times min\{x - nq_n(x),\ n(q_n(x)+1) - x\}} \in [0, 1]$$
  where $\lambda > 1$, $\beta > 0$ and $q_n(x) = \left\lfloor \frac{x}{n} \right\rfloor$ (quotient of euclidian division of x by n)
- Intuition behind this reward function (TODO : figure instead) :
  - The closer we are to a multiple of n, the greater the reward
  - For all x, $nq_n(x) \leq x < n(q_n(x) + 1)$: x is always between two multiples of n
  - Among these two multiples, we choose the closest one to x, then we compute the distance $d = min\{x - nq_n(x),\ n(q_n(x) + 1) - x\}$.
  - To transform the distance into a reward, we just do $\lambda^{-\beta \times d}$, $\lambda > 1$ and $\beta > 0$.

I - Problem to solve: Example

Generate (randomly) **positive** multiples $x_1, \ldots, x_m$ of a positive integer n, with each x_i being of length less than or equal to $\ell > 2$.

Attempt 2

- State space

$$\mathcal{X} = \{0, ..., 10^{\ell} - 1\}$$

- Reward function ($n = 5$, $\lambda = 2$ and $\beta = 1$) :

$$R(x) = \begin{cases} 1 & \text{if } x = 5k \\ 2^{-1} & \text{if } x = 5k+1 \text{ or } x = 5k+4 = 5(k+1)-1 \\ 2^{-2} & \text{if } x = 5k+2 \text{ or } x = 5k+3 = 5(k+1)-2 \end{cases} \qquad (k \in \mathbb{N})$$

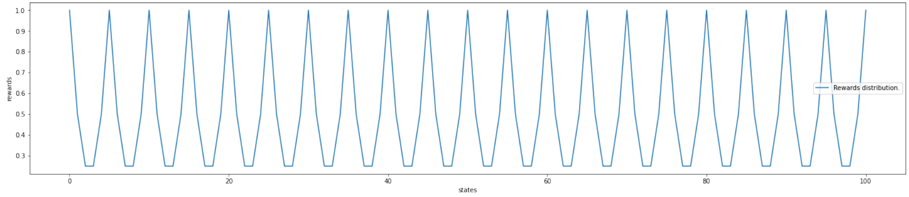


Figure: Reward distribution

I - Example : Generate (randomly) multiples of n

- Reward function ($n = 5$, $\lambda = 2$ and $\beta = 1$) :

$$R(x) = \begin{cases} 1 & \text{if } x = 5k \\ 2^{-1} & \text{if } x = 5k+1 \text{ or } x = 5k+4 = 5(k+1)-1 \\ 2^{-2} & \text{if } x = 5k+2 \text{ or } x = 5k+3 = 5(k+1)-2 \end{cases} \qquad (k \in \mathbb{N})$$

- Canonical partition function

$$Z = \sum_{x \in \mathcal{X}} R(x) = \sum_{x=0}^{10^{\ell}-1} R(x)$$

- Probability distribution

$$\pi(x) = \frac{R(x)}{Z} = \begin{cases} 1/Z & \text{if } x = 5k \\ 1/2Z & \text{if } x = 5k+1 \text{ or } x = 5k+4 \\ 1/4Z & \text{if } x = 5k+2 \text{ or } x = 5k+3 \end{cases} \quad (k \in \mathbb{N})$$

I - Problem to solve : Why is this important?

Machine Learning (Ubiquitous ...)

- Stochastic GD
- Weight initialization, Random kitchen sinks
- Bayesian inference, Variational Inference
- ...

Statistical physics (Ising model, ...)

$$R(x) = e^{-\frac{E(x)}{k_B T}}$$

- $E(x)$ is the energy (Hamiltonian) of the microstate x (of the physical system)
- T the temperature
- k_B the Boltzmann constant
- $\pi(x) = \frac{R(x)}{Z}$ the Boltzmann factor at x (probability for the system to be present in the microstate x)

II - Inverse Transform Sampling

Proposition (How it works?)

- **cdf** : $F : \mathbb{R} \to [0, 1]$
- **Uniform RNG** : $U \sim \mathcal{U}([0, 1])$
- The random variable $X = F^{-1}(U)$ is distributed as F

$$\mathbb{P}\left[X \leq x\right] = \mathbb{P}\left[F^{-1}(U) \leq x\right] = \mathbb{P}\left[U \leq F(x)\right] = F(x) \text{ since } F(x) \in [0, 1]$$

Example : Exponential distribution

$$F(x) = 1 - e^{-\lambda x} \implies F^{-1}(U) = -\frac{ln(1-U)}{\lambda}$$

Advantages & Limits

- Advantages : cheaper (when F^{-1} is available)
- Limits : quantile function F^{-1}
  In some cases (normal distribution ...), we don't even have a formula for F

II - Inverse Transform Sampling

Alternatives

- Find an estimate of F^{-1}
  Normal distribution : when x is large, $F(x)$ behaves like $\tilde{F}(x) = 1 - e^{-x^2/2}$ whose density is $\tilde{f}(x) = xe^{-x^2/2}$ and the inverse is $\tilde{F}^{-1}(u) = \sqrt{ln(1-u)}$.
  Rayleigh's distribution
- Root search inversion for continuous distributions
  Sometimes we have or can approximate F better than F^{-1} : for a given U, look for X such that $F(X) - U = 0$.
  Binary search, Newton-Raphson, regula falsi, secant method, Brent-Dekker ...
- Inversion for discrete distribution
  $p(x_i) = P[X = x_i]$ for $i = 0, \dots, k-1$ and $F(x) = \sum_{x_i \leq x} p(x_i)$
  Generate U, find $I = min\{i \mid F(x_i) \geq U\}$ and return x_I.
  Sequential search, binary search, Index search, ...
- Acceptance-Rejection Method
  - Most important technique after inversion
  - Can provide an effective solution when inversion is too difficult or costly

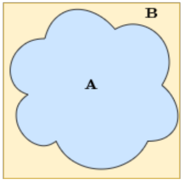
III - Acceptance-Rejection Method

Proposition

- We want to generate a point uniformly in a set $\mathcal{A} \subset \mathbb{R}^d$
- We choose a "simpler" set $\mathcal{B}$ such that $\mathcal{A} \subset \mathcal{B}$
- We generate independent points in $\mathcal{B}$, and retain the first one that falls in $\mathcal{A}$
- The retained point follows the uniform distribution in $\mathcal{A}$

$$\text{For } \mathcal{D} \subset \mathcal{A},\ \mathbb{P}\left[\mathbf{X} \in \mathcal{D} | \mathbf{X} \in \mathcal{A}\right] = \frac{\mathbb{P}\left[\mathbf{X} \in \mathcal{D} \cap \mathcal{A}\right]}{\mathbb{P}\left[\mathbf{X} \in \mathcal{A}\right]} = \frac{vol(\mathcal{D})/vol(\mathcal{B})}{vol(\mathcal{A})/vol(\mathcal{B})} = \frac{vol(\mathcal{D})}{vol(\mathcal{A})}$$

Thus the distribution of $\mathbf{X}$ conditional on $\mathbf{X} \in A$ is uniform in $\mathcal{A}$



III - Acceptance-Rejection Method

- We want to generate $\mathbf{X}$ according to a density $f : \mathbb{R} \to \mathbb{R}_+$
- The surface under f is:

$$S(f) = \left\{(x, y) \in \mathbb{R}^2 : 0 \leq y \leq f(x)\right\}$$

- Proposition : If $(\mathbf{X}, \mathbf{Y})$ is uniform on $S(f)$, then $\mathbf{X}$ has the density f

$$(\mathbf{X}, \mathbf{Y}) \sim \mathcal{U}\left(S(f)\right)$$
$$\Longrightarrow \mathbb{P}[\mathbf{X} \leq x] = Area\left(\{(x, y) \in S(f) : z \leq x\}\right) = \int_{-\infty}^{x} f(z)dz = F(x)$$

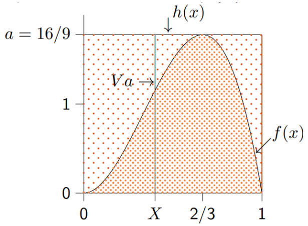
- Idea: Choose a simple surface $\mathcal{B}$ which contains $S(f)$, then generate (X, Y) uniformly in $\mathcal{B}$. If $(X, Y) \in S(f)$, it is OK, otherwise we start again.

III - Acceptance-Rejection Method

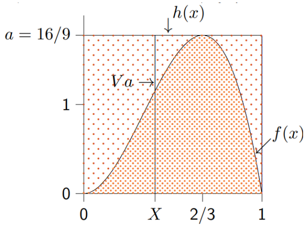
- $X \sim Beta(3,2)$
- cdf $f(x) = 12x^2(1-x) \cdot \mathbb{1}_{[0,1]}(x)$
- $\arg\max_{x\in[0,1]} f(x) = 2/3$
- $a = f(2/3) = 16/9$

$$\mathcal{B} = \{(x,y) : 0 \leq x \leq 1, 0 \leq y \leq 16/9\}$$

- To generate (X, Y) in $\mathcal{B}$, we generate two independent uniforms U and V, and return $(X, Y) = (U, aV)$



III - Acceptance-Rejection Method



- To generate (X, Y) in $\mathcal{B}$, we generate two independent uniforms U and V : $(X, Y) = (U, aV)$
- The probability that the point is in $S(f)$ is $\frac{\int_0^1 f(x)dx}{a} = 1/a = 9/16$
- The expected number of points (X, Y) to be generated is $a = 16/9$
  This is because the number of iterations needed to successfully generate a candidate follows a geometric distribution with success probability $1/a$.
  $\Longrightarrow$ we can tune a : we want $vol(\mathcal{B})$ as small as possible

III - Acceptance-Rejection Method

- To generate X according to density f, we choose another density g and a constant $a \geq 1$ such that

$$f(x) \leq h(x) = ag(x), \forall x$$

  and such that it is easy to generate X according to g
- We apply the rejection method with $\mathcal{A} = S(f)$ and $\mathcal{B}$ the surface under h

$$\mathcal{B} = S(h) = \left\{(x, y) \in \mathbb{R}^2 : 0 \leq y \leq h(x)\right\}$$

Algorithm 1: Acceptance-Rejection Method

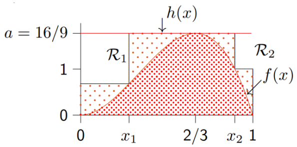
```
1 repeat
2     Generate $X$ according to $g$ and $U \sim \mathcal{U}([0,1])$, independently
3 until $\underline{U \cdot h(X) \leq f(X)}$
4 Return $X$
```

III - Acceptance-Rejection Method

Algorithm 2: Acceptance-Rejection Method

```
1 repeat
2     Generate $X$ according to $g$ and $U \sim \mathcal{U}([0,1])$, independently
3 until $\underline{U \cdot h(X) \leq f(X)}$
4 Return $X$
```

- At each loop turn, the probability of accepting X is $1/a$
- The number of loop turns before acceptance is a geometric r.v. of parameter $p = 1/a$.
- The average number of loop turns per r.v. is $1/p = a$.
- Therefore, we want $a \geq 1$ as small as possible.
- Tradeoff between decreasing a and keeping g simple



Example

- $X \sim Beta(3, 2)$
- To reduce a, we can take the hat function

$$h(x) = \begin{cases} f(x_1) & \text{if } x < x_1 \\ 16/9 & \text{if } x_1 \le x \le x_2 \\ f(x_2) & \text{if } x_2 < x \end{cases} \quad (0 < x_1 < 2/3 < x_2 < 1)$$

- The area under h is minimized by taking $x_1 = 0.281023$ and $x_2 = 0.89538$. It is then reduced from 1.77778 to 1.38997.
- The inverse cdf G^{-1} of g is piecewise linear.
- Why not take a piecewise linear function h instead of a piecewise constant? Computing G^{-1} would require square roots (...) : we would lose efficiency.

III - Acceptance-Rejection Method

Limits

In many problems, Z, the denominator of π is (too) hard to compute. .

- Bayesian inference : $\pi(\theta|x) = \frac{\pi(\theta)p(x|\theta)}{p(x)}$
  - $\pi(\theta)p(x|\theta)$ is easy to compute
  - but $p(x) = \int \pi(\theta)p(x|\theta)d\theta$ can be very difficult or impossible to compute
  - So we need to be able to sample from the posterior $\pi(\theta|x)$ just by knowing $\pi(\theta)p(x|\theta)$, ie the prior $\pi(\theta)$ and the likelihood $p(x|\theta)$.
- Statistical physics (e.g. Ising model)
  - We only know $R(x) = e^{-\frac{E(x)}{k_B T}}$, but $Z \propto \int_{\mathcal{X}} R(x)dx$ is often difficult to compute

Alternatives

MCMC, Variational Inference (Bayes), GflowNets, etc

It may also be tempting to see R as a function to be optimized and try directly the gradient-based approaches, but note that R is derivative-free : evaluations do not give gradient information.

IV - MCMC

- MCMC methods try to find a (reversible) Markov chain of transition matrix P such that π is its stationary distribution

$$\pi P = \pi, \quad \pi(x) = \frac{R(x)}{Z} \; \forall x$$

- MCMC trick: the constraint remains valid if we multiply the two sides by a constant

$$\pi P = \pi \iff Z\pi P = Z\pi \iff RP = R$$

- If such a Markov chain is found, it will be enough to simulate it up to some step $n \to \infty$:
    - simulate a random walk of infinite length on the chain, $\pi_{n+1} = \pi_n P$ with $n \to \infty$ and take the last elements obtained
    - the first steps is generaly call the burn-in phase
- Given the difficulty of finding π in practice, MCMC methods find alternatives to simulate a random walk without explicitly having π, but just R. Metropolis-Hasting, Gibbs sampling, Metropolis-adjusted Langevin, ...

IV - MCMC : Metropolis-Hasting

- We want to sample from a distribution $\pi(x) = R(x)/Z$, $R(x) > 0$ and Z unknow
- We define a Markov transition matrix Q on $\mathcal{X}$: $0 \leq Q \leq 1$ and $\sum_{x' \in \mathcal{X}} Q(x'|x) = 1 \ \forall x \in \mathcal{X}$
- Let $x_0 \in \mathcal{X}$ the initial state
- We then perform the following two steps repeatedly:
  - $$x = x_t, \ generate \ x' \sim Q(.|x)$$
  - $$x_{t+1} = \begin{cases} x' & \text{with probability } \alpha(x'|x) = min\left\{\frac{R(x')\cdot Q(x|x')}{R(x)\cdot Q(x'|x)}, 1\right\} \\ x & \text{otherwise.} \end{cases}$$
- Note that α depends only on R, not π.
  So we don't need π in the algorithm, only $R = \pi Z$.
- Markov transition matrix (it works?) :

$$P(x'|x) = \alpha(x'|x)Q(x'|x)$$

IV - MCMC : Metropolis-Hasting

- Detailed balance equation :

$$\begin{aligned}
\forall\ (x, x') \in \mathcal{X}^2, \quad \pi(x)P(x'|x) &= \pi(x)\ min\left\{\frac{Z\pi(x')\ .\ Q(x|x')}{Z\pi(x)\ .\ Q(x'|x)}, 1\right\}Q(x'|x) \\
&= min\Big\{\pi(x')Q(x|x'), \pi(x)Q(x'|x)\Big\} \\
&= \pi(x')\ min\left\{1, \frac{Z\pi(x)\ .\ Q(x'|x)}{Z\pi(x')\ .\ Q(x|x')}\right\}Q(x|x') \\
&= \pi(x')\alpha(x|x')Q(x|x') \\
&= \pi(x')P(x|x')
\end{aligned}$$

- So the markov chain of transition matrix P is reversible with stationary distribution π :

$$\pi P = \pi$$

$$\forall\ x \in \mathcal{X},\ \sum_{x'} \pi(x')P(x|x') = \sum_{x'} \pi(x)P(x'|x) = \pi(x)\sum_{x'} P(x'|x) = \pi(x)$$

IV - MCMC : Metropolis-Hasting

- Markov transition matrix (it works? Yes) :

$$P(x'|x) = \alpha(x'|x)Q(x'|x)$$

- Detailed balance equation :

$$\forall \; (x, x') \in \mathcal{X}^2, \;\; \pi(x)P(x'|x) = \pi(x')P(x|x')$$

- So the markov chain of transition matrix P is reversible with stationary distribution π :

$$\pi P = \pi$$

- This condition, although sufficient, is not necessary for π to be a stationary distribution
- It is also necessary for the Markov Chain to be ergotic (i.e. that there exists n such that $P^n > 0$) to guarantee that π is stationary: in this case, π is the unique stationary distribution of the Markov Chain

IV - MCMC : Metropolis-Hasting

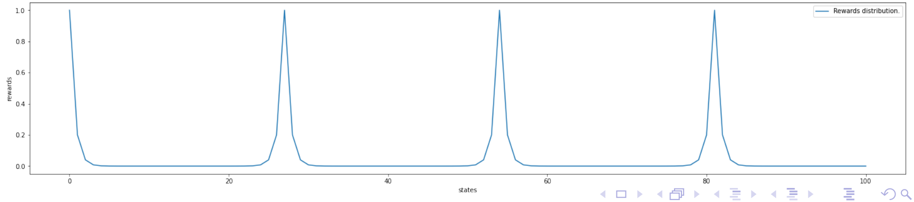
Example : generating multiples of $n = 27$

- We can consider an isotropic Gaussian distribution as the proposal distribution :

$$Q(.|x) \sim \mathcal{N}(x, \sigma^2)$$

- The algorithm will be very sensitive to the value of σ^2 chosen.
- If σ^2 is too small, the algorithm may turn indefinitely around its starting point (because of the desert that exists between the modes of R).
- We can also choose a uniform proposal distribution :

$$Q(x'|x) = \frac{1}{|\mathcal{X}|} \ \forall \ (x, x') \in \mathcal{X}^2$$



IV - MCMC

Limits

- Generally too expensive
- Generally do only local exploration
  If π has several modes separated by deserts of probabilities, these algorithms can get stuck in the same regions during the random walk
- The space $\mathcal{X}$ on which the reward function is defined is very complex in many problems : set of molecules, the set of sentences, set of images, etc
  - Sampling requires very complex algorithms or models to build objects to be evaluated
  - An algorithm that tries to successively build the objects and test them (like MCMC methods) would be very expensive

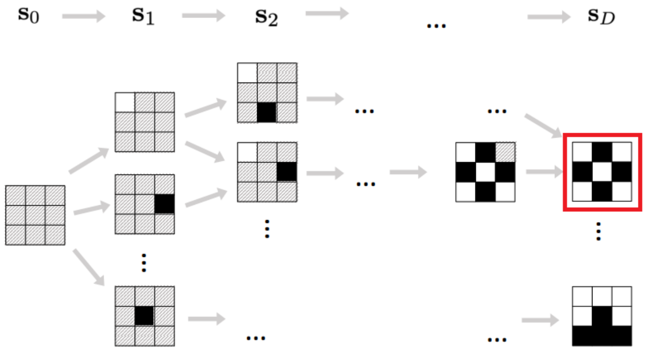
Alternatives

Variational Inference (Bayes), GflowNets, etc

V - GFlowNets

Definition

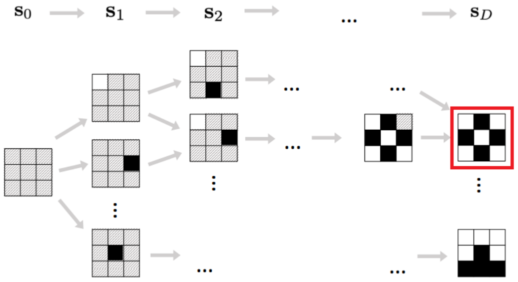
GFlowNets are generative models that learn a stochastic policy that iteratively constructs the sampled object through a sequence of simpler steps, such that the probability of generating an object is proportional to its reward



V - GFlowNets : Interesting when the problem have some properties

Objects can be constructed sequentially

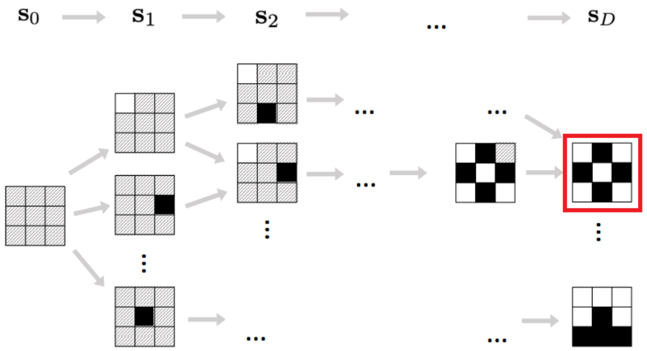
A molecule for example can be built by combining several atoms, a sentence can be constructed by adding each time a word after the sentence already constructed. A binary image can be generated by choosing at each step a pixel position, and then choosing the color (0 or 1) to assign to it



V - GFlowNets : Interesting when the problem have some properties

The same object can be built (and destroyed) in several different ways

We are dealing with a DAG (Directed Acyclic Graph)



V - GFlowNets : Interesting when the problem some properties

The same object can be built (and destroyed) in several different ways

We are dealing with a DAG (Directed Acyclic Graph)

The DAG can in many cases be reduced to a simple tree

- For example, considering that the final sentence **[BOS] a b c [EOS]** has been constructed in an autoregressive way
- **[BOS]** : *Beginning Of Sentence*
- **[EOS]** : *End Of Sentence*
- It is obvious that the sentence that led to this state is **[BOS] a b c**
- And the one that led to **[BOS] a b c** is **[BOS] a b**
- ...
- ... **[BOS] a**
- ... **[BOS]** (initial state)

V - GFlowNets: Interesting when the problem have some properties

The reward function has several modes (local maximums)

- Example: multiples of a number n of length less than or equal to $\ell > 2$
- I assume here that the numbers are constructed autoregressively starting from an initial state (like **[BOS]**) and adding each time a digit between 0 and 9 at the end of the already constructed number
- The process continues until a special symbol is generated (like **[EOS]**) or until the number is of length ℓ
- If we have instead a neural network that directly returns a number (like the generator of a GAN), then we cannot speak of a notion of flow

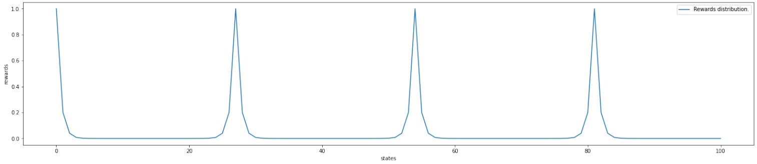


Figure: $R(x) = \lambda^{-\beta \times min\{x - n*q_n(x),\ n*(q_n(x)+1) - x\}}$

The oracle (the real reward function in other words) is difficult to call

- In the case of the multiples of n above, the analytical form of the reward function is known. Moreover, with a little mathematics one can find all x such that $R(x) = r, \ r \in \mathbb{R}^+$
- But let's imagine the case where we have to find a molecule that kills the given virus, and that has other properties like its time of action (if the molecule kills the virus very slowly a sick person can take the drug, but die before it does its effects), that has few side effects, that is easy to manufacture... All these properties can be translated into a reward function, except that here, being given a new molecule, the specialists have to go to the laboratory, do clinical trials (which take days or weeks, which are financially expensive ...).
- An alternative in this case (and that GFlowNets knows very well how to exploit), is to train on the molecules already available (but which are not as satisfactory as we wish), a model able to estimate the reward of new molecules. In the case of covid-19, a dataset can be made of : Moderna, Pfizer-BioNTech, AstraZeneca, Janssen (Johnson & Johnson) ...
- This reward function (less expensive), can be used to evaluate molecules very quickly.

V - GFlowNets : Interesting when the problem have some properties

Reinforcement Learning?

Note that Reinforcement Learning algorithms can also be used to solve this type of problems, but the maximization of expected return in this algorithms is generally achieved by putting all the probability mass of the state-action policy on the highest-return sequence of actions. GflowNets solves the problem by turning R into a generative policy which samples with a probability proportional to R.

V - GFlowNets : DAG

Directed graph

Tuple $(\mathcal{S}, \mathcal{A})$

- $\mathcal{S}$: set of states
- $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$: directed edges
- Elements of $\mathcal{A}$ are denoted $s \rightarrow s'$ and called **edges** or **transitions**.

For each element of $s \in \mathcal{S}$, we define two sets :

- the set of its parents : $Par(s) = \{s' \mid s' \rightarrow s \in \mathcal{A}\}$
- and the set of its children : $Child(s) = \{s' \mid s \rightarrow s' \in \mathcal{A}\}$

Trajectory

Sequence $\tau = (s_1, \dots, s_n)$ of elements of $\mathcal{S}$ such that every transition $s_t \rightarrow s_{t+1} \in \mathcal{A}$:

- $s \in \tau$ means that s is in the trajectory τ, i.e., $\exists\ t \in \{1, \dots, n\}\ /\ s_t = s$
- $s \rightarrow s' \in \tau$ means that $\exists t \in \{1, \dots, n\}\ /\ s_t = s, s_{t+1} = s'$

Directed Acyclic Graph : DAG

Directed graph in which there is no trajectory $\tau = (s_1, \ldots, s_n)$ satisfying $s_n = s_1$, besides trajectories composed of one state only.

Source & Sink

For DAGs considered here, we have two special states :

- the **source/initial state** s_0, $Par(s_0) = \emptyset$
- the **sink/final state** s_f, $Child(s_f) = \emptyset$

Depending on the problem we can have many of these states. In this case, a special initial (resp. final) state can be created which is connected to all initial states (resp. to which all final states are connected)

Complete trajectory

- Trajectory that starts at s_0 and ends at s_f. Hereafter, the complete trajectories $\tau = (s_0, s_1, \ldots, s_n, s_f)$ will be noted simply $\tau = (s_0, s_1, \ldots, s_n)$
- Set of complete trajectories : $\mathcal{T} = \{(s_0, \ldots, s_n) \mid s_n \in \mathcal{X}\}$

Problem to solve

Let $\mathcal{X}$ be the set of final states, and let define on $\mathcal{X}$ a positive function R
How to define on $(\mathcal{S}, \mathcal{A})$ a generative model for which the probability $\pi(x)$ of generating an element $x \in \mathcal{X}$ is proportional to $R(x)$, ie $\pi(x) = \frac{R(x)}{\sum_{x \in \mathcal{X}} R(x)}$?

Solution

GflowNets define a function $F : \mathcal{T} \mapsto \mathbb{R}_+$, called flow :

- which must respect the reward matching condition :

$$R(x) = \sum_{\tau=(s_0,\ldots,s_n) | s_n = x} F(\tau)$$

- which must be Markovian, i.e. there must exist distributions $P_F(.|s)$ over the children $Child(s)$ of every non-terminal state s, and a constant Z, such that for any complete trajectory $\tau = (s_0, \ldots, s_n)$ we have

$$P(\tau) = P_F(\tau) = P_F(s_n|s_{n-1})P_F(s_{n-1}|s_{n-2})\ldots P_F(s_1|s_0) = \frac{F(\tau)}{Z}$$

Total flow : partition function

$$Z = \sum_{\tau \in \mathcal{T}} F(\tau)$$

As in statistical physics, Z captures how the rewards/probabilities are distributed among different individual final states / complete trajectories.

Forward policy

$P_F(s_{t+1}|s_t)$ is called a forward policy : by using it to construct a trajectory $\tau = (s_0, \dots, s_n)$, the probability that $s_n = x \in \mathcal{X}$ is proportional to $R(x)$. Why?

Backward policy : Backwards transition probability P_B

There must exist distributions $P_B(.|s)$ over the parent $Par(s)$ of every non-initial state s such that for any terminal state x we have :

$$P(\tau = (s_0, \dots, s_n) \mid s_n = x) = P_B(s_0|s_1) \dots P_B(s_{n-2}|s_{n-1}) P_B(s_{n-1}|s_n = x)$$

Given $x \in \mathcal{X}$, P_B can be used to sample a trajectory leading from s_0 to x.

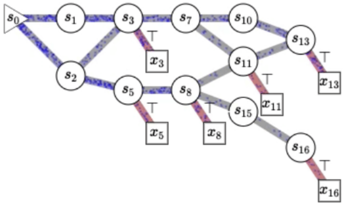
State flow and edge flow

The flow through a state is the sum of the flows of the complete trajectories passing through that state

$$F(s) = \sum_{\tau \in \mathcal{T}} \mathbb{1}_{s \in \tau} F(\tau)$$

The flow $F(s \to s')$ through an edge $s \to s'$ is called an edge flow

$$F(s \to s') = \sum_{\tau \in \mathcal{T}} \mathbb{1}_{s \to s' \in \tau} F(\tau)$$



- $F(s_0) = F(s_f) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z$ since $\forall \tau \in \mathcal{T}$, $s_0 \in \tau$ and $s_f \in \tau$
- In-flow for non-initial states :

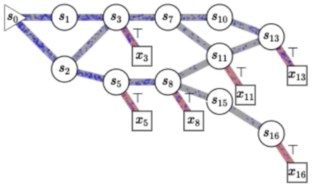
$$F_{in}(s) = \sum_{s'' \in Par(s)} F(s'' \rightarrow s)$$

- Out-flow for non-terminal states :

$$F_{out}(s) = \sum_{s' \in Child(s)} F(s \rightarrow s')$$

- We can also show that

$$P_F(s'|s) = \frac{F(s \rightarrow s')}{F(s)} \text{ and } P_B(s|s') = \frac{F(s \rightarrow s')}{F(s')}$$



Problem to solve

Let $\mathcal{X}$ be the set of final states, and let define on $\mathcal{X}$ a positive function R.
How to define on $(\mathcal{S}, \mathcal{A})$ a generative model for which the probability $\pi(x)$ of generating an element $x \in \mathcal{X}$ is proportional to $R(x)$, ie $\pi(x) = \frac{R(x)}{\sum_{x \in \mathcal{X}} R(x)}$?

Solution

Using P_B, generate complete trajectories $\tau = (s_0, \dots, s_n = x)$ such that

$$F_{out}(s_0) = Z \text{ (redundancy : the other two conditions imply this one)}$$

$$F(s_t) = F_{in}(s_t) = F_{out}(s_t) \ \forall \ t \in 1, ..., n$$
$$F_{out}(s_n) = F_{out}(x) = R(x)$$

If this objective is satisfied, we will have $\pi(x) = \frac{R(x)}{Z} \ \forall \ x \in \ \mathcal{X}$. Why?

Proposition

Let $\pi(s)$ denote the probability of visiting state s when starting at s_0 and following $P_F(\cdot|\cdot)$. Then $\pi(s) = \frac{F(s)}{F(s_0)}$

Proof : by induction.

- Trivially true for the root s_0 : $\pi(s_0) = 1$
- For an intermediate state s, if the statement is true for all $s' \in Par(s)$, then we have :

$$\begin{aligned}
\pi(s) = \sum_{s' \in Par(s)} \pi(s') P_F(s' \to s) &= \sum_{s' \in Par(s)} \frac{F(s')}{F(s_0)} \frac{F(s' \to s)}{F(s')} \\
&= \frac{1}{F(s_0)} \sum_{s' \in Par(s)} F(s' \to s) \\
&= \frac{F(s)}{F(s_0)}
\end{aligned}$$

□

Proposition

Let $\pi(s)$ denote the probability of visiting state s when starting at s_0 and following $P_F(\cdot|\cdot)$. Then $\pi(s) = \frac{F(s)}{F(s_0)}$

Theorem (Solution)

Using P_B, generate complete trajectories $\tau = (s_0, \ldots, s_n = x)$ such that

$$F(s_t) = F_{in}(s_t) = F_{out}(s_t) \; \forall \; t \in 1, ..., n$$

$$F_{in}(s_n) = F_{out}(s_n) = R(x)$$

If this objective is satisfied, we will have $\pi(x) = \frac{R(x)}{Z} \; \forall \; x \in \mathcal{X}$

Proof.

Applying the proposition to any final state x, we have $\pi(x) = \frac{F(x)}{F(s_0)} = \frac{R(x)}{Z}$ $\square$

V - GFlowNets : Training

Flow matching objective (FM)

- A model $F_\theta(s, s')$ with learnable parameters θ approximates the edge flows $F(s \to s')$.
- The corresponding forward policy is given by

$$P_F(s'|s, \theta) = \frac{F_\theta(s, s')}{F(s)} = \frac{F_\theta(s, s')}{\sum_{s'' \in Child(s)} F_\theta(s \to s'')}$$

The parameters are trained to minimize the error in the flow matching constraint for all non-initial nodes s :

$$\mathcal{L}_{FM}(s) = \begin{cases} \left(log \frac{\sum_{s'' \to s} F_\theta(s'', s)}{R(s)} \right)^2 & \text{if } s \text{ terminal} \\ \left(log \frac{\sum_{s'' \to s} F_\theta(s'', s)}{\sum_{s \to s'} F_\theta(s, s')} \right)^2 & \text{else.} \end{cases}$$

Flow matching objective (FM)

- The parameters are trained to minimize the error in the flow matching constraint for all non-initial nodes s :

$$\mathcal{L}_{FM}(s) = \begin{cases} \left(log \frac{\sum_{s'' \to s} F_\theta(s'', s)}{R(s)} \right)^2 & \text{if } s \text{ terminal} \\ \left(log \frac{\sum_{s'' \to s} F_\theta(s'', s)}{\sum_{s \to s'} F_\theta(s, s')} \right)^2 & \text{else.} \end{cases}$$

- This objective is optimized for nonterminal states s and terminal states x from trajectories sampled from a training policy π_θ, a tempered (higher temperature) version of $P_F(.|s,\theta)$ which also helps exploration during training

$$\pi_\theta(.|s) = log\left(e^{\frac{P_F(.|s,\theta)}{T}} \right)$$

- The parameters are updated with stochastic gradient

$$\mathbb{E}_{\tau=(s_0,\dots,s_n)\sim\pi_\theta} \nabla_\theta \left[\sum_{t=1}^{n} \mathcal{L}_{FM}(s_t) \right]$$

V - GFlowNets : Training

Flow matching objective

- From a computational point of view, the FM loss is difficult to compute when $|Par(s)|$ is large, for many s, because it requires, being at a given non-initial state s, to enumerate all its parents, to evaluate the model F_θ on each of the parents, and to consider only the flows leading to s
- Credit assignment : the states visited earlier in the trajectory have more credits than those visited later

V - GFlowNets : Training

Detailed balance objective (DB)

$$P_F(s'|s) = \frac{F(s \to s')}{F(s)} \text{ and } P_B(s|s') = \frac{F(s \to s')}{F(s')} \implies F(s)P_F(s'|s) = F(s')P_B(s|s')$$

- This is the detailed balance constraint
- It is a bit similar to the reversibility condition of Markov chains
- $DB \ \wedge \ F(x) = R(x) \ \forall x \in \mathcal{X} \implies FM$

A neural network model with parameters θ has input s and three outputs:

- an estimated state flow $F_\theta(s)$
- an estimated distribution over children $P_B(.|s,\theta)$
- and an estimated distribution over parents $P_F(.|s,\theta)$

Detailed balance objective (DB)

- DB : $F(s)P_F(s'|s) = F(s')P_B(s|s')$
- $DB \ \wedge \ F(x) = R(x) \ \forall x \in \mathcal{X} \implies FM$
- A neural network model with parameters θ has input s and three outputs: $F_\theta(s)$, $P_B(.|s,\theta)$ and $P_F(.|s,\theta)$
- The error in the detailed balance constraint is optimized on actions $s \rightarrow s'$ between nodes seen along trajectories.

$$\mathcal{L}_{DB}(s,s') = \begin{cases} \left(log \frac{F_\theta(s)P_F(s'|s;\theta)}{R(s')P_B(s|s';\theta)} \right)^2 & \text{if } s' \text{ terminal} \\ \left(log \frac{F_\theta(s)P_F(s'|s;\theta)}{F_\theta(s')P_B(s|s';\theta)} \right)^2 & \text{else.} \end{cases}$$

- Similarly to flow matching, the parameters are updated with stochastic gradient

$$\mathbb{E}_{\tau=(s_0,\ldots,s_n)\sim\pi_\theta} \nabla_\theta \left[\sum_{t=1}^{n} \mathcal{L}_{DB}(s_{t-1}, s_t) \right]$$

V - GFlowNets : Training

Trajectory Balance objective (TB)

For any terminal state x we have :

$$\begin{aligned}\prod_{t=1}^{n} P_B(s_{t-1}|s_t) &= P(\tau = (s_0, \ldots, s_n) \mid s_n = x) \\ &= \frac{P(\tau = (s_0, \ldots, s_n = x))}{P(s_n = x)} \\ &= \frac{\prod_{t=1}^{n} P_F(s_t|s_{t-1})}{\frac{F(x)}{Z}}\end{aligned}$$

This gives the trajectory balance constraint for any complete trajectory $\tau = (s_0, ..., s_n = x)$:

$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t),\ Z = F(s_0),\ F(x) = R(x)$$

V - GFlowNets : Training

Trajectory Balance objective (TB)

- A model with parameters θ outputs
  - estimated forward policy $P_F(.|s, \theta)$
  - and backward policy $P_B(.|s, \theta)$ for states s
  - as well as a global scalar Z_θ estimating $F(s_0)$.
- For a trajectory $\tau = (s_0, ..., s_n = x)$, the trajectory balance constraint defines the trajectory loss ($F(x)$ is replaced by $R(x)$ because we want the two quantities to be equal) :

$$\mathcal{L}_{TB}(\tau = (s_0, ..., s_n = x)) = \left(log \frac{Z_\theta \prod_{t=1}^{n} P_F(s_t|s_{t-1};\theta)}{R(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t;\theta)} \right)^2$$

- The trajectory loss is updated along trajectories sampled from π_θ, a tempered version of $P_F(.|., \theta)$, i.e., with stochastic gradient

$$\mathbb{E}_{\tau \sim \pi_\theta} \nabla_\theta \mathcal{L}_{TB}(\tau)$$

Conclusion

- Outperforms MCMC, PPO (proximal policy optimization) ... on drug discovery ...

VI - More resources

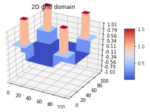
Tutorial

I recommend my tutorial [a] which shows in detail how to implement the three losses

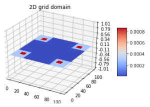
- GflowNets on Hypergrid environment : below, example with
- GflowNets for Image Generation, and for GAN
- GflowNets for Sequence Generation

[a] `https://github.com/Tikquuss/GflowNets_Tutorial`

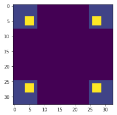
$$R(x) = R_0 + R_1 \prod_{i=1}^{n} \mathbb{1}(0.25 < |x_i/H - 0.5|) + R_2 \prod_{i=1}^{n} \mathbb{1}(0.3 < |x_i/H - 0.5| < 0.4)$$



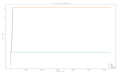
(a) $R_0 <<< R_1 < R_2$



(b)



(c) GflowNets



(d) reward

VI - More resources

More details about GflowNets

- https://milayb.notion.site/GFlowNet-Tutorial-919dcf0a0f0c4e978916a2f509938b00
- To go in depth: *GflowNets foundations* paper (Bengio et al., 2021) or *Trajectory Balance* paper (Malkin et al., 2022) (very pedagogical paper).

- RVs and Stochastic Processes Generation (and more) : Pierre L'Ecuyer, Stochastic Simulation and Monte Carlo Methods, IFT-6561, DIRO, UdeM. His book, a masterclass, is not yet public. But if you ask for access he will send it to you. The part about the rejection method and some pictures are inspired from this book
- For Variational Bayes, I recomment this paper : A practical tutorial on Variational Bayes (Tran et al., 2021)
- MCMC and Bayesian Modeling, 2017, Martin Haugh, Columbia University [a]

[a] http://www.columbia.edu/~mh2078/MachineLearningORFE/MCMC_Bayes.pdf

[1] Yoshua Bengio et al. "GFlowNet Foundations". In: CoRR abs/2111.09266 (2021). arXiv: 2111.09266. URL: https://arxiv.org/abs/2111.09266.

[2] Nikolay Malkin et al. "Trajectory Balance: Improved Credit Assignment in GFlowNets". In: CoRR abs/2201.13259 (2022). arXiv: 2201.13259. URL: https://arxiv.org/abs/2201.13259.

[3] Minh-Ngoc Tran et al. "A practical tutorial on Variational Bayes". In: arXiv preprint arXiv: Arxiv-2103.01327 (2021).