The GFlowNets and Amortized Marginalization Tutorial

(see here for the more comprehensive GFlowNet tutorial, although the stuff below might be useful to first provide a gist of core ideas behind the ability of GFlowNets to learn to sample and marginalize over distributions for which exactly doing these tasks is intractable)

1. Simple MSE Criterion to Amortize an Intractable Expectation

Consider a set of intractable expectations that we would like to approximate (for any x)

$$S(x) = \sum_y p(y|x)R(x,y)$$

where y is a rich variable taking an exponential number of values and we know how to sample from $p(y|x)$.

We could train a neural net with input x, stochastic target $R(x,y)$ and MSE loss

$$L = \left(\widehat{S}(x) - R(x,y)\right)^2$$

where $y \sim p(y|x)$ and training estimator $\widehat{S}$ with parameters θ and the stochastic gradients $\frac{\partial L}{\partial \theta}$ would make $\widehat{S}$ converge to S if it has enough capacity and is trained long enough.

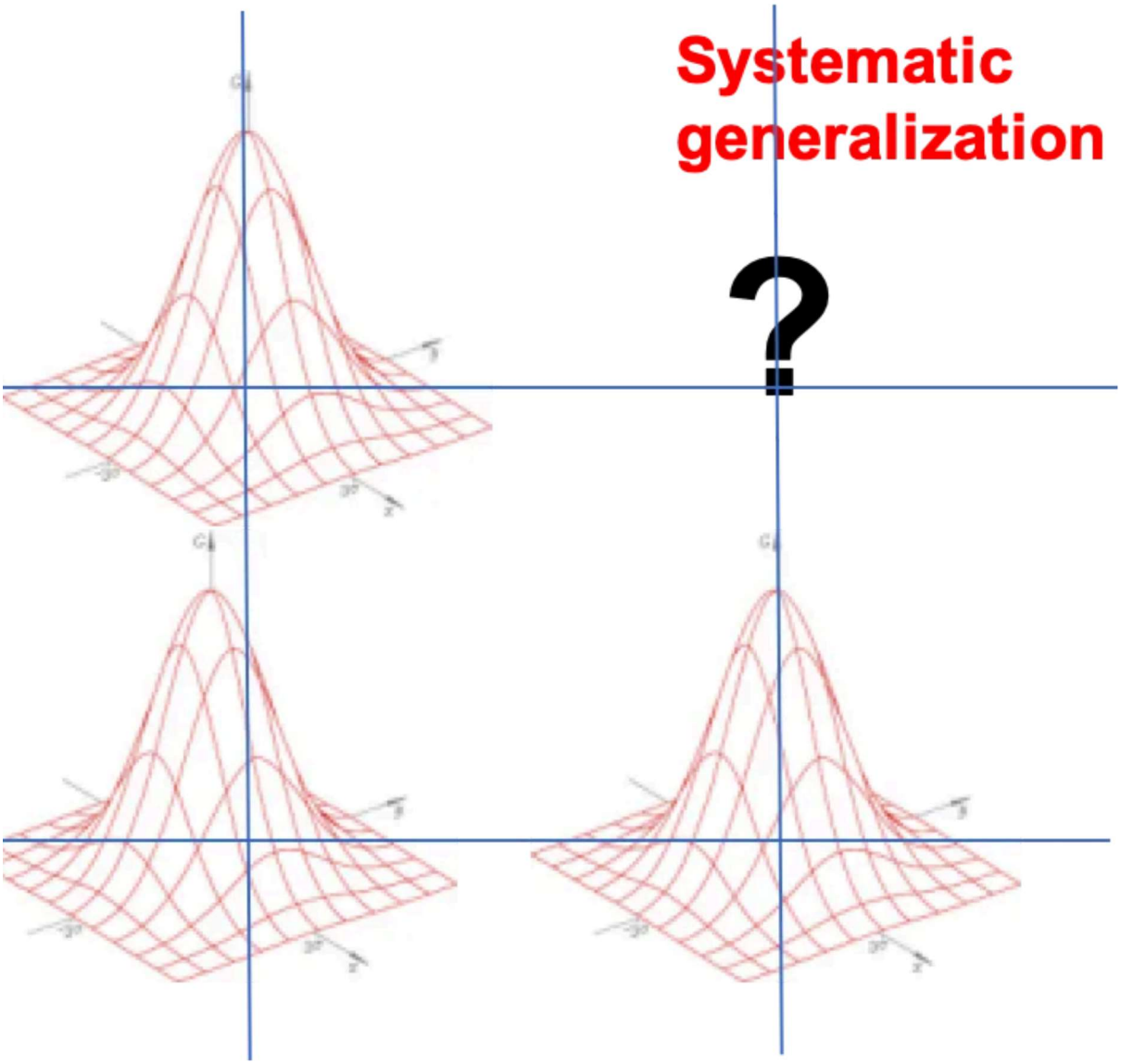
For any new x, we would then have an amortized estimator $\widehat{S}(x)$ which in one pass through the network would give us an approximation of the intractable sum. We can consider this an efficient alternative to doing a Monte-Carlo approximation

$$\widehat{S}_{MC}(x) = \text{mean}_{y \sim p(y|x)} R(x,y)$$

which would require a potentially large number of samples and computations of $R(x,y)$ for each x at run-time, especially if $p(y|x)$ is a rich multimodal distribution (for which just a few samples does not give us a good estimator of the expectation).

Besides the advantage of faster run-time, a crucial potential advantage of the amortized version is that it could benefit from **generalizable structure** in the product $p(y|x)R(x,y)$: if observing a training set of $(x, R(x,y))$ pairs can allow us to generalize to new pairs, then we may not need to train $\widehat{S}$ with an exponential number of examples before it captures that generalizable structure and provides good answers (i.e., $E_{Y|x}[R(x,Y)]$) on new x's. The ability to generalize like this is actually what explains the remarkable success of machine learning (and in particular deep learning) in vast set of modern AI applications.

When we do not have a $p(y|x)$ that we can sample from easily, we can in principle use MCMC methods, that form chains of samples of y whose distributions converge to the desired $p(y|x)$, and where the next sample is generally obtained from the previous one by a small stochastic change that favours increases in $p(y|x)$. Unfortunately, when the modes of $p(y|x)$ occupy a small volume (i.e. throwing darts does not find them) and these modes are well-separated (by low-probability regions), especially in high dimension, it tends to take exponential time to mix from one mode to another. However, this is leaving money on the table: the attempts $(x, y, R(x,y))$ actually contain information one could use, using ML, to guess where the yet unseen modes might be, given where some of the modes (those already observed), as illustrated below.



The above figure illustrates that if we have already the three modes in red, we may guess the presence of a 4th mode on the top-right grid point, because the first three modes seem to align on a grid. The existence of such generalization structure is why amortized ML samplers can potentially do much better MCMC samplers.

2. GFN Criterion to Obtain a Sampler and Estimate Intractable Sums

Let us consider the situation where we do not have a handy $p(y|x)$ and our objective is just to approximate a set of intractable sums (for any x)

$$S(x) = \sum_y R(x, y)$$

where we have the constraint that $R(x,y) \geq 0$ and $S(x) > 0$. This may be useful to estimate a normalization constant for energy-based models or Bayesian posteriors (where y=parameters and x=data). Hence we may also be interested in the sampling policy

$$\pi(y|x) = \frac{R(x,y)}{S(x)} \propto R(x,y).$$

Now, the idea of GFN losses is based on the notion of a constraint that we would like to be true:

$$\forall (x,y): \quad \pi(y|x)S(x) = R(x,y). \tag{1}$$

We can define estimators $\widehat{\pi}$ and $\widehat{S}$ and train them with a loss such as

$$L(x,y) = \left(\widehat{\pi}(y|x)\widehat{S}(x) - R(x,y)\right)^2$$

or with an interpretation of R as unnormalized probabilities that we want well calibrated in the log-domain,

$$L(x,y) = \left(\log(\widehat{\pi}(y|x)\widehat{S}(x)) - \log R(x,y)\right)^2$$

where (x,y) are sampled from a training distribution $\tilde{p}(x,y)$ that has full support. It can then be shown (main GFN theorem, see NeurIPS paper and GFN Foundations paper) that with $\widehat{\pi}$ and $\widehat{S}$ with enough capacity and trained for long enough they both converge to their desired value:

$$\begin{aligned}\widehat{S}(x) &\to S(x) \\ \widehat{\pi}(y|x) &\to \pi(y|x) \propto R(x,y).\end{aligned}$$

3. Marginalizing over Compositional Random Variables

To make the notion of intractable sum more concrete, it is good to think of y (and potentially x as well) as a compositional object, like a subset of variable-value pairs or a graph. To make this more concrete, we think of compositional objects being constructed sequentially through a series of steps where a new piece is inserted at each step. The sampling policy π then samples these steps (which we call **actions**) and after each step we get a **partially constructed object** s which we call a GFN **state**. The sequence of such states and actions forms a GFN **trajectory**. In the basic GFN framework, these transitions are deterministic (i.e. a state and an action determine exactly the next state) because they are not happening in an external environment but are part of the internal computation of a sampler. In addition, the GFN math as it currently stands (and it can probably be extended) assumes that each step is constructive, i.e., we cannot return to the same partially constructed object s twice, which means that the set of all possible trajectories forms a directed acyclic graph (DAG). Because the transitions are deterministic, we can specify a trajectory τ with a sequence of states (we could have equally chosen an initial state and a sequence of actions). A special action also needs to be defined to declare that a state s can be generated as a fully constructed object y. The policy $\pi(y|x)$ is now specified by a forward transition distribution $P_F(s|s')$ which specifies how to generate each constructive step, and we are interested in parametrizing and learning this P_F.

Multiple Parent States

Besides the sequential nature of the generative process for y, an interesting complication is that there may be many ways (in fact exponentially many trajectories) to construct y from some starting point and context x. This means that a partially construct object, i.e., a state s, may have multiple parents s' for which an action a exists that leads to s. Otherwise, the DAG is a tree, which makes the math and calculations much simpler. But when it is not a tree, it turns out to be convenient to consider and parametrize a backward transition probability function $P_B(s'|s)$ which is consistent with that DAG and the associated P_F. The constraint in Eq. (1) can be reformulated in several ways, in particular what is called the detailed balance constraint:

$$\forall (s, s') : \quad P_F(s|s')F(s') = P_B(s'|s)F(s) \tag{2}$$

where $F(s)$ is called the flow at state s and plays a role similar to $S(x)$ above, i.e., it is an intractable sum, and there is a starting state $s_0 = x$ from which a trajectory is initiated, as well as a constraint that the flow into a terminal state $s = y$ equals $R(x, y)$. Similarly to the simpler case above, this can be turned into a training loss that we want to minimize, but now over all (x, τ) pairs or over all (s, s') pairs. We also note that the initial flow $F(s_0) = S(x)$, i.e., the initial flow provides us with the normalizing constant:

$$F(s_0) = S(x) = \sum_y R(x, y).$$

4. Implications for Bayesian ML

Let us consider the special case where $y = \theta$ is a latent parameter, i.e., in a Bayesian setting, and $x = D$ is the available data. Then we can define the reward function

$$R(D, \theta) = P(\theta)P(D|\theta)$$

from the parameter prior $P(\theta)$ and the data likelihood $P(D|\theta)$. Training a GFN with a policy $\pi(\theta|D)$ and initial flow $S(D)$ provides us with an approximate sampler for the posterior over parameters given data as well as an estimator of the normalizing constant of the Bayesian posterior. Hence, we have used amortization to turn a tractable quantity (prior times likelihood) into estimators of these intractable quantities. We get a fast sampler for the posterior with no need for a Markov chain going through a large number of candidate samples, from which we can generate many samples $\theta|D$ in an iid fashion that are likely to visit the larger modes of the posterior.

To make computations more ML-friendly while training the GFN, we can note that the GFN squared loss objectives naturally lend themselves to the case where the reward or log-reward is stochastic and an unbiased estimator of the true reward. For example, we can typically decompose the overall dataset log-likelihood $\log P(D|\theta)$ into a sum of per-example or per-minibatch terms, and we can introduce a multiplicative correction to account for the prior $P(\theta)$:

$$\begin{aligned}\log \widehat{R}(Z, \theta) &= \log P(\theta) + |D| \log P(Z|\theta) \\ E_Z[\log \widehat{R}(Z, \theta)] &= \log P(\theta) + \sum_{Z \in D} \log P(Z|\theta) = \log R(D, \theta).\end{aligned}$$

This makes it possible to train the GFN posterior estimator using SGD on single examples or minibatches, which is the state-of-the-art to train deep nets.

In addition to training a GFN to represent the parameter posterior $Q(\theta|D)$, one could train it to estimate jointly the predictive posterior $Q(y|x, X, Y)$, where $D = (X, Y)$ is formed of (x, y) pairs, X contains the x-part of D and Y the y-part. This could be achieved with the same reward $R(D, \theta) = P(\theta)P(D|\theta)$ but with a generative policy $Q(Y, \theta|X)$ that first samples Y given X via $Q(Y|X)$ and then samples the parameters from $Q(\theta|X, Y)$. In the case of exchangeable examples, the (x, y) pairs in D are iid given θ, so that

$$Q(Y|X) = \prod_{i=1}^{|D|} Q(y_i|y_1^{i-1}, x_1^i)$$

where $y_1^n = (y_1, y_2, \ldots, y_n)$ and we are training (with the factors above) a predictive posterior $Q(y|x, D)$ that can predict the output y for a new experiment x given all the past experimental results D.

5. Implications for Causal ML

A causal model can be seen as a family of distributions indexed by interventions, where all the distributions in the family share the same parameters specifying causal dependencies between random variables (but differ on which variables were intervened and what were the values imposed on these variables). One of the main challenges in learning a causal model is that discovering the causal graph, a directed acyclic graph (DAG) with one node per random variable and a directed edge between a random variable and one of its direct causes. One problem is that the number of such DAGs is super-exponential in the number of variables. Another is identifiability: if we only observe data from one member of that family (typically the default of no intervention), then in general there are many DAGs that are compatible with the data (they are collectively called the Markov Equivalence Class or MEC), even as the amount of data goes to infinity. On the other hand, we may have the opportunity to observe the outcomes of many interventions (also called experimental results), and then the ambiguity about the correct causal graph should decrease. The MEC is interesting but does not cover the case where the amount of data is finite and/or we observe different experiments. A more general description of the ambiguity regarding the causal graph is given by the Bayesian posterior on the causal graph, given the available information (data and possibly other sources of information which may constrain the causal graph). However, this is a highly multimodal distribution in a discrete space, which makes it difficult to represent and even learn it. GFNs have been used to represent and learn that distribution, taking advantage of the simple ideas in the previous section and the abilities of GFNs to learn distributions over graphs: see Deleu et al, UAI'2022.

6. Implications for Experimental Design

An appealing theoretical framework for defining the objective of an experiment is that of information gain: how much information about a random variable of interest can we expect to gain through the experiment? Experimental design could be driven by this information gain as as reward. Note that we can replace the word "experiment" by "action" to start thinking about exploration and information seeking in reinforcement learning, and such a framework also applies to in the context of active learning or interactive learning more generally. In general, information gain may not be the only effect of an action, though. For example there could be risks involved. We can however incorporate a simple notion of cost or budget by considering the information gain per unit of cost incurred.

Information gain can be measured in principle by the mutual information between the outcome of an experiment (a random variable since the experiment has not taken