GFlowNets and Hierarchical Variational Inference

GFlowNets can be seen as extending the already rich family of **amortized variational inference** methods, more specifically amortized hierarchical variational inference, where there are several latent variables $s_1, s_2, \ldots$ and they are hierarchically organized so that when we generate an object x we start by sampling s_1, then $s_2|s_1$, then $s_3|s_2$, etc and then convert the last s_{n-1} into $x = s_n$ with a model $P(s_1, s_2, \ldots, s_{n-1}, x) = P(x|s_{n-1})P(s_{n-1}|s_{n-2}) \ldots P(s_2|s_1)$. As discussed in [7] and [9], the sequence $\tau = s_1, \ldots, s_{n-1}, x$ corresponds to the GFlowNet trajectory τ, with $P_F(\tau) = P(x|s_{n-1}) \ldots P(s_2|s_1)$. As with other variational inference approaches one also learns an inference machine $Q(\tau|x)$ which corresponds to $P_B(\tau|x)$ in GFlowNets. With variational method we also have a marginal data distribution (which we could denote $Q(x)$) that with GFlowNets corresponds to the normalized reward function $R(x)/Z$ that can be queried as needed by the training procedure (unlike a fixed dataset). Another difference is that variational methods are trying minimize the reverse KL divergence between Q and P (or using importance sampling, the forward KL) whereas GFlowNets are trained with a diversity of losses, e.g. corresponding to something like $(\log Q(\tau) - \log P(\tau))^2$ with Trajectory Balance, that open the possibility of offline training (with trajectories τ sampled from a distribution different from the online samples from Q). In [9], we show that when a GFlowNet is trained with trajectory balance on-policy, the expected gradient is the same as with variational inference and the reverse KL, but the variance is different (and the trajectory balance gradient is equivalent to using a variance reduction trick, compared with regular variational inference). The typical variational inference objective (the ELBO or reverse-KL) leads to mode-following (focussing on one mode) and the forward KL leads to mean-following (overly conservative, sampling too broadly) and annoying variance when implemented with importance sampling. Instead the off-policy GFlowNet objectives (e.g., with a tempered version of P_F as training policy) seem to strike a different balance and tend to recover more of the modes without the down-side of the forward-KL variational inference variants (mean-following and high variance gradients). Another difference is that GFlowNets can learn flows and conditional flows, which correspond to marginalized quantities, as discussed above.

GFlowNets and RL

The basic objective of RL policies is to find trajectories that maximize return, whereas the constructive policies of GFlowNets are trained so they sample terminal states that match some desired unnormalized probability function (which is the reward function for GFlowNet). Hence they solve different problems. Similarly, the value function in RL estimates an expected sum of downstream rewards (which can be positive or negative), after going through a state s or a transition $s \rightarrow s'$. In contrast, the flow function of GFlowNets at state s or transition $s \rightarrow s'$ estimates the fraction of the sum of all downstream rewards (rewards must be non-negative here) attributable to going through s or $s \rightarrow s'$.

Finally, as discussed in [1], one can modify the training objective in RL - by adding an entropy maximization regularizer on the policy - to make the policy sample in proportion to the return, like a GFlowNet. However, that only works if there is only one path leading to each state (the GFlowNet DAG is a tree), whereas the GFlowNet training procedures work in all cases (whether the DAG is a tree or not).

It is amusing to note that RL with Entropy Regularization frames the control problem as an inference problem, whereas GFlowNets frame the inference problem as a control problem.

Another important difference between the typical RL setting and the GFlowNet setting is that the GFlowNet policy (corresponding to P_F above) only allows "constructive" actions, which means that the graph of possible state transitions forms a DAG, i.e., one cannot go back to a previously visited state. This made sense in the original paper [1], where the GFlowNet policy constructively samples a graph by adding edges. In contrast, the action-space of RL does not have that constraint. However, the GFlowNet state-space can be augmented (e.g., by a time-stamp) to guarantee the DAG-ness of state-space, and it is possible to create sequences of states that include both forward transitions (sampling from P_F) and backward transitions (sampling from P_B) and this is in fact how [4] defines a particular MCMC process based on GFlowNets and allowing large jumps.

Jointly Training the Sampler and an Energy Function

In [4], we have shown how we can jointly train the GFlowNet sampler and its target reward function (which is an energy function in disguise: $R(x) = e^{-\mathcal{E}(x)}$) from a dataset. We use the classical maximum likelihood gradient estimator for energy functions, which requires sampling from the current model defined by the energy function, but we use the GFlowNet to produce those samples. We find even better results if we use a contrastive divergence variant, whereby the negative samples are one MCMC step away from a training example, but each step of the the Markov chain is obtained by resampling parts of x with a few backward steps and a few forward steps in the GFlowNet.

Another option is to consider the parameters of the energy function as random variables over which the GFlowNet can also sample. In that case, there are no energy function parameters to be learned by maximum likelihood: instead the GFlowNet can be trained to sample from the Bayesian posterior directly (getting a high reward when it chooses energy function parameters that have a high prior and give a high probability to the data). A first step in that direction is [5], where the GFlowNet learns to sample from the Bayesian posterior over causal graphs (or graphical model graphs), given the training data. Another option, in the case where we can compute $P(x|z)$ for x observed data and z latent variables sampled by the GFlowNet (approximately from $P(z|x)$), is the GFlowNet-EM procedure [14], which generalizes the variational inference framework of GFlowNets to approach maximum likelihood training in the context where there are high-dimensional structured latent variables and possibly a rich multimodal distribution for the posterior distribution $P(z|x)$.

GFlowNets with Latent Variables

Can GFlowNets handle latent variables? Yes, but we can think of several ways.

The trajectory used to construct a terminal state could involve sampling intermediate variables that are not mentioned in the reward function. Think about a sentence and its parse tree and a GFlowNet that constructively builds both. The reward function could simply reflect the probability of occurrence of the words and ignore the parse tree. However, the parametrization and conditional independence assumptions involved in generating a (sentence,parse tree) pair could make it easier to capture dependencies when exploiting latent variables to construct the sentence (for example, in a traditional generative grammar, given a parent variable p in the tree, the children and descendants of p in the tree are assumed to be independent).

Alternatively, the energy function could be learned (as in [4]) and involve both observed variables and latent variables. Even the parameters of the energy function could be considered latent variables and be sampled by the GFlowNet, as outlined above. Finally, if we use the GFlowNet to sample from a Bayesian posterior, it can do so jointly over the latent variables and the parameters of the energy function.