RL with pathwise and score-function gradients

Matias Alvo
Advisors: Dan Russo, Yash Kanoria

---

1. Problem Setup

We consider a reinforcement learning problem over T periods. In each period t, the decision-maker(DM) observes a state S_t, picks a action a_t from a feasible set $\mathcal{A}_t$, observes a cost c_t and the state transitions to state S_{t+1}. We will consider problems where the actions in each period are continuous, but where there is a large fixed penalty for strictly positive actions (modeling, *e.g.*, large fixed setup costs in a manufacturing problem). We will consider the Hindsight RL problem framework, so we assume that there is an observable and exogenous driving noise ξ_t in each period, and that the cost and transition functions are known, and are deterministic functions of a (state, action, exogenous noise) triplet.

Note that the cost function c_t has a large discontinuity at 0. This, in principle, should preclude the application of HDPO. Moreover, a differentiable approximation of the cost function seems problematic, given that the gradient of cost wrt actions near 0 may not approximately reflect the true gradient. For instance, if we use very "steep" functions near 0, the gradient of the fixed cost will be very large for actions sufficiently near from 0, dissuading an agent from increasing the action. On the other hand, score-function (REINFORCE) estimators can directly be used as gradient estimators, but likely with a high variance. Can we approach this type of problems in a convenient way so that we can take advantage of the low-variance gradient estimate given by pathwise gradients?

[MA: NOTE: I believe using this hybrid-action formulation could help address other problems as well. The general idea is to consider settings with a finite/small number of 'jumps' in the costs/transitions/action, with this jumps being known a priori (I am not pretending to model *e.g.*, that in an auction with unknown valuations, the reserve price induces discontinuities at an unkown

value). We would then model these 'jumps' via discrete actions, and control the value within the specific valid range with continuous actions. I leave some examples here]

- Feasible actions set is a disconnected set (and some regions are not 'point')
  - —A supplier might only accept orders above a certain threshold.
  - —Investment vehicle requires a minimum investment amount
  - —Loans might be available in brackets due to that *e.g.*, Small loans (1,000–100,000) are managed by "retail banking" and large loans (500,000+) are managed by "wholesale banking".
- Cost is discontinuous wrt actions
  - —Vendors manage many products. Orders are transported together in one truck/container. [MA: suggested in a previous meeting]
  - —Supplier needs to send an additional truck after hitting certain quantities
  - —Retailer might offer "bulk" discounts (or free shipping) when hitting certain thresholds
- Transition is discontinuous wrt actions
  - —The lead time offered by a supplier depends on the number of units ordered
  - —If money/resource is part of the state and we are maximizing value, "bulk" discounts create discontinuities

Model Description

[MA: I am currently posing the model formulation in a way that facilitates explaining some of our proposed approaches, though will probably re-write for future versions. Also, I am considering the case where there are large fixed costs, but I believe the spirit of this idea should extend to the previously mentioned use cases by having one possible value for the stochastic action for each possible 'region/range' of the action/transition/cost, and having one separate deterministic action for each 'region/range'.]

In this section, we will present a formulation of the model in which the actions a_t in each period are decomposed into a stochastic component a_t^S and a deterministic component a_t^D. Stochastic components a_t^S will typically be binary variables modeling, *e.g.*, whether we set–up a machine for manufacturing items in the current period. Meanwhile, a_t^D will typically represent a continuous variable modeling *e.g.*, how much to produce *if* a machine is set-up. The idea behind this action decomposition is that deterministic actions will "flow" through the stochastic computation graph Schulman et al. (2015) in a deterministic manner, allowing us to compute pathwise gradients. On the other hand, stochastic components a_t^S will, in principle, necessitate score-function type of estimators (we will later propose approximations that might allow us to obtain patwhwise gradients).

The problem is detailed as follows:

- S_t: State at time t.
- $a_t = a_t^S \cdot a_t^D$: Action at time t, decomposed as:
  - a_t^S: Stochastic component of the action, sampled from a distribution parameterized by ϕ based on the state S_t.
  - a_t^D: Deterministic component of the action, determined by θ as a function of S_t.
- ϕ: Parameter vector mapping S_t to the distribution of a_t^S.
- θ: Parameter vector mapping S_t to a_t^D.
- $\pi_{\theta,\phi}$: policy parameterized by θ and ϕ
- ξ_t: Exogenous noise at time t, influencing both the cost and state transition.
- $c_t(S_t, a_t, \xi_t)$: Cost incurred at time t.
  - For training the agent, we will consider that the cost function takes the form $c_t(S_t, a_t^S, a_t^D, \xi_t)$. This is, it depends separately on a_t^S and a_t^D. We do this with the goal that the agent can correctly tell apart the fixed costs from the variable costs. Typically, we will consider costs that can be decomposed as $D_t^S \cdot a_t^S + c_t^D(S_t, a_t^D, \xi_t)$, representing a large fixed costs and a variable one.
- $f(S_t, a_t, \xi_t)$: Deterministic transition function governing the next state:

$$S_{t+1} = f(S_t, a_t, \xi_t).$$

Objective

The objective is to find a policy $\pi_{\theta,\phi}$ that minimizes total expected cost, given by

$$J(\theta, \phi) = \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t, a_t^S, a_t^D, \xi_t)\right].$$

The problem, given a parameterized class of functions, is given by

$$\min_{\theta \in \Theta, \phi \in \Phi} J(\theta, \phi)$$

2. Solution approaches

We will propose several approaches to solve this problem. The first approach relies on utilizing a score-function estimator to update the parameters ϕ that map states to distributions over stochastic actions. Meanwhile, to update parameters θ that map states to deterministic actions, it uses a combination of score-function and pathwise gradients estimates, disentangling the effect that these actions have on the distribution from which stochastic actions are sampled, and their deterministic effect on costs. To derive the gradients, we will visualize the computations performed using stochastic computation graphs Schulman et al. (2015).

The second methodology builds on the previous approach by approximating the binary actions via the Gumbel-softmax trick Jang et al. (2016), Maddison et al. (2016). At a high-level, this trick enables a "smooth" approximation of binary variables by injecting Gumbel noise to a deterministic output, dividing by a temperature parameter τ and applying a softmax operator. The value of the temperature controls how 'discrete' or 'continuous' the output is. Moreover, τ can be annealed during training, so that the agent gradually learns how to output a binary action. It is important to note that this trick allows for applying the 'reparameterization trick', which allows one to obtain pathwise gradients from this smooth approximation. All in all, the application of the Gumbel-softmax trick in this setting would allow obtaining pathwise gradients for all parameters.

We will likely consider two naive benchmarks for comparison. The first one directly enforces a smooth approximation of the cost function c_t (via 'steep' functions near 0), and without decomposing into a stochastic and deterministic component. We can train utilizing pathwise gradients. We expect this approach to encounter issues, given that the gradients near 0 actions will not properly approximate the marginal cost of actions.

A second benchmark will be a direct application of a model-free approach (such as PPO) to the model that does nos decomposes a_t. [MA: It is actually not obvious to me how to apply this. Should we discretize actions? Or first sample whether to "act" or 'not to act' according to a Bernoulli(θ), with trainable θ, and then sample from a Gaussian (with trainable parameters) if 'act'?]

2.1. Hybrid-action RL

As previously mentioned, this approach relies on obtaining gradient estimators that are composed of score-function and pathwise gradients. The stochastic computation graph for a problem with $T = 3$ periods can be visualized in Figure 1. Following Theorem 1 in Schulman et al. (2015), we can compute the gradient $\nabla_{(\theta,\phi)} J(\theta, \phi)$.

As an example, we show the computation of the gradient of the objective $J(\theta, \phi)$ with respect to the parameters θ. The key points to keep in mind while computing this quantity are the following:

- The **score function gradient** captures the indirect effect of θ on the cost through its impact on the distribution of the stochastic actions a_t^S.
- The **pathwise derivative gradient** captures the direct effect of θ on the deterministic actions a_t^D and their influence on the cost and state transitions.

The gradient can be decomposed into two parts: the score function gradient and the pathwise derivative gradient.

Score Function Gradient The score function gradient accounts for the influence of θ on the probability distribution of the stochastic nodes a_t^S:

$$\left.\frac{\partial J}{\partial \theta}\right|_{\text{SF}} = \sum_{t=1}^{T} \mathbb{E}\left[\frac{\partial \log p(a_t^S \mid S_t, \phi)}{\partial \theta} \cdot Q_t\right], \tag{1}$$

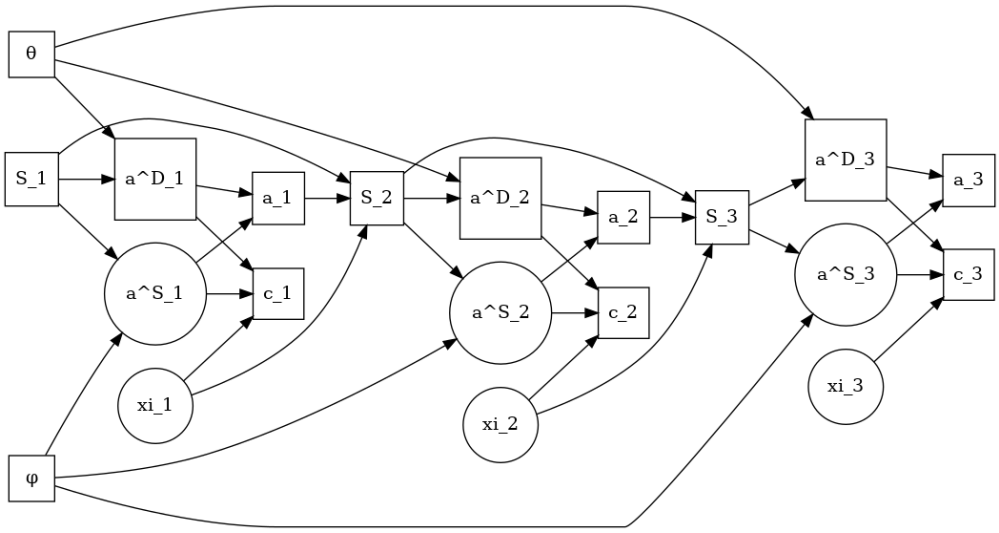

Figure 1 **Stochastic Computation Graph for $T = 3$ periods. Circles represent stochastic nodes, while square nodes represent deterministic nodes.**

where $Q_t = \sum_{u=t}^{T} c_u$. We note that as the stochastic action a_t^S is indirectly influenced by θ through all previous deterministic actions a_u^D $(u < t)$, the term $\frac{\partial \log p(a_t^S | S_t, \phi)}{\partial \theta}$ will be decomposed into a sum (see Section 2.1.1).

Pathwise Derivative Gradient The pathwise derivative gradient accounts for the influence of θ on the deterministic components a_t^D and indirectly on the cost through state transitions:

$$\left.\frac{\partial J}{\partial \theta}\right|_{\text{PD}} = \sum_{t=1}^{T} \mathbb{E}\left[\frac{\partial c_t}{\partial a_t^D} \cdot \frac{\partial a_t^D}{\partial \theta}\right]. \tag{2}$$

We note that the deterministic action a_t^D depends on θ both directly and indirectly through all previous deterministic actions a_u^D $(u < t)$, so the term $\frac{\partial a_t^D}{\partial \theta}$ will decompose into a sum (see Section 2.1.1).

Combined Gradient The total gradient of the objective $J(\theta, \phi)$ with respect to θ is:

$$\frac{\partial J}{\partial \theta} = \sum_{t=1}^{T} \mathbb{E}\left[\frac{\partial \log p(a_t^S \mid S_t, \phi)}{\partial \theta} \cdot Q_t\right] + \sum_{t=1}^{T} \mathbb{E}\left[\frac{\partial c_t}{\partial a_t^D} \cdot \frac{\partial a_t^D}{\partial \theta}\right]. \tag{3}$$

We note that in PyTorch we will be able to automatically obtain the pathwise gradients. However, in order to obtain the score-function part of the gradient, we will need to compute (in the forward pass) the desired quantities to differentiate.

2.1.1. Detailed Gradient Decomposition Gradient of $\log p(a_t^S \mid S_t, \phi)$ with Respect to θ

The stochastic action a_t^S is indirectly influenced by θ through all previous deterministic actions a_u^D $(u < t)$ and the resulting states S_t. The gradient can be written as:

$$\frac{\partial \log p(a_t^S \mid S_t, \phi)}{\partial \theta} = \sum_{u=1}^{t-1} \frac{\partial \log p(a_t^S \mid S_t, \phi)}{\partial S_t} \cdot \frac{\partial S_t}{\partial a_u^D} \cdot \frac{\partial a_u^D}{\partial \theta}. \tag{4}$$

Here:

- $\frac{\partial \log p(a_t^S|S_t,\phi)}{\partial S_t}$: Sensitivity of the stochastic policy's log-probability with respect to the state.
- $\frac{\partial S_t}{\partial a_u^D}$: Sensitivity of the state S_t to the deterministic action a_u^D, determined by the transition function.
- $\frac{\partial a_u^D}{\partial \theta}$: Sensitivity of the deterministic action a_u^D to the parameter θ.

Gradient of a_t^D with Respect to θ The deterministic action a_t^D depends on θ both directly and indirectly through all previous deterministic actions a_u^D $(u < t)$ and the resulting states S_t. The gradient is:

$$\frac{\partial a_t^D}{\partial \theta} = \frac{\partial a_t^D}{\partial \theta}\bigg|_{\text{direct}} + \sum_{u=1}^{t-1} \frac{\partial a_t^D}{\partial S_t} \cdot \frac{\partial S_t}{\partial a_u^D} \cdot \frac{\partial a_u^D}{\partial \theta}. \tag{5}$$

Here:

- $\frac{\partial a_t^D}{\partial \theta}\Big|_{\text{direct}}$: Direct dependence of a_t^D on θ.
- $\frac{\partial a_t^D}{\partial S_t}$: Sensitivity of a_t^D to the state S_t.
- $\frac{\partial S_t}{\partial a_u^D}$: Sensitivity of the state S_t to the deterministic action a_u^D.

Full Gradient of the Cost Function The full gradient of the expected cost $J = \mathbb{E}[C]$ with respect to θ can be decomposed into two parts: the score function gradient and the pathwise derivative gradient.

Score Function Gradient:

$$\frac{\partial J}{\partial \theta}\bigg|_{\text{SF}} = \sum_{t=1}^{T} \mathbb{E}\left[\sum_{u=1}^{t-1} \frac{\partial \log p(a_t^S \mid S_t, \phi)}{\partial S_t} \cdot \frac{\partial S_t}{\partial a_u^D} \cdot \frac{\partial a_u^D}{\partial \theta} \cdot Q_t\right], \tag{6}$$

where Q_t represents the cumulative effect of costs influenced by a_t^S.

Pathwise Derivative Gradient:

$$\frac{\partial J}{\partial \theta}\bigg|_{\text{PD}} = \sum_{t=1}^{T} \mathbb{E}\left[\frac{\partial c_t}{\partial a_t^D} \cdot \frac{\partial a_t^D}{\partial \theta}\right]. \tag{7}$$

Combined Gradient: The total gradient is:

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial \theta}\bigg|_{\text{SF}} + \frac{\partial J}{\partial \theta}\bigg|_{\text{PD}}. \tag{8}$$

2.2. Gumbel-Softmax Hybrid-action RL

In this approach, we aim to approximate the binary actions a_t^S using the Gumbel-Softmax reparameterization trick, which provides a continuous relaxation of the discrete binary decisions. This allows us to apply the reparameterization trick for gradient-based optimization while maintaining a valid distribution for the binary action. The Gumbel-Softmax distribution is controlled by a temperature parameter τ, which determines the smoothness of the relaxation. By tuning τ, we can control how close the approximation is to a discrete binary decision.

At each time step t, the action a_t^S is sampled from the Gumbel-Softmax distribution:

$$a_t^S = \text{Gumbel-Softmax}(z_t, \tau), \tag{9}$$

where z_t represents the logits of the binary action space, and τ is the temperature parameter. The Gumbel-Softmax function is defined as:

$$\text{Gumbel-Softmax}(z_t, \tau) = \frac{\exp\left(\frac{z_t + g_t}{\tau}\right)}{\sum_{j=1}^{2} \exp\left(\frac{z_t^j + g_t^j}{\tau}\right)}, \tag{10}$$

where g_t is the Gumbel noise, and τ controls the sharpness of the distribution. As τ approaches zero, the distribution becomes more deterministic, with the outcome approaching a one-hot vector corresponding to a discrete binary action. Conversely, as τ increases, the distribution becomes smoother, allowing for more continuous-valued approximations of the binary actions.

Training Procedure The Gumbel-Softmax trick allows for continuous relaxation of binary actions, making the process end-to-end differentiable. Therefore, libraries like PyTorch can automatically compute gradients in the backwards pass.

During training, τ can either be fixed or annealed. When τ is fixed, it serves as a hyperparameter that controls the degree of smoothness of the action distribution throughout the training process. For example, a smaller value of τ would push the approximation towards a discrete decision, while larger values might be chosen to provide a smoother approximation during training, which may help in certain cases to avoid vanishing gradients.

Alternatively, τ can be annealed during training to gradually encourage discrete actions as training progresses. A typical annealing schedule could be defined as:

$$\tau_t = \frac{\tau_0}{1 + \beta t}, \tag{11}$$

where τ_0 is the initial temperature and β is the decay rate. This schedule encourages the model to begin with a smoother approximation and transition towards more discrete decisions as training progresses. However, it is also possible that a fixed τ may suffice, especially in cases where the task benefits from a more continuous relaxation.

In order to generate discrete action during training while maintaining pathwise differentiability, we might use the straight-through trick. This would entail using a hard-max in the forward pass, while using the following gradient during the backwards pass:

$$\hat{a}_t^S = \text{round}(a_t^S) \quad \text{(straight-through estimator)}. \tag{12}$$

Inference At inference time, we need to generate truly discrete binary actions. Since τ is typically not zero at inference, directly applying the Gumbel-Softmax sample may not yield perfectly discrete outcomes. In this case, we can use a common strategy to approximate discrete binary actions by rounding the output of the Gumbel-Softmax:

$$\hat{a}_t^S = \mathbb{I}\left(a_t^S > 0.5\right), \tag{13}$$

where $\mathbb{I}$ is the indicator function that outputs 1 if $a_t^S > 0.5$ and 0 otherwise. This ensures that the output is a binary action, though the quality of the approximation may degrade if τ is not sufficiently small. To improve inference quality, τ should be tuned such that the action distribution is close enough to a hard binary decision. In some cases, this may require careful tuning or a small post-processing threshold to guarantee discrete outputs in practice.

Gradient Estimation The gradients of the objective function with respect to the parameters are computed automatically via automatic differentiation tools, such as those in PyTorch, which can propagate gradients through the Gumbel-Softmax function.

References

Jang E, Gu S, Poole B (2016) Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* .

Maddison CJ, Mnih A, Teh YW (2016) The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* .

Schulman J, Heess N, Weber T, Abbeel P (2015) Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems* 28.