Probability Functional Descent: A Unifying Perspective on GANs, Variational Inference & Reinforcement Learning

Casey Chu <caseychu@stanford.edu> Jose Blanchet Peter Glynn

Optimizing probability functionals

Let $J : \mathcal{P}(X) \to \mathbb{R}$ be a function, where X is a topological space, and $\mathcal{P}(X)$ denotes the set of all probability distributions on X. We call such functions **probability functionals**.

The goal is to find a probability distribution $\mu \in \mathcal{P}(X)$ that minimizes $J(\mu)$.

A unifying perspective

Many modern machine learning problems can be formulated this way:

- **Generative models.** Let ν_0 be a data distribution that we want to mimic with μ. This can be framed as minimizing a divergence to ν_0. In this case, we let
$$J_{\text{GM}}(\mu) = D(\mu \,||\, \nu_0),$$
where $D(\cdot\,||\,\cdot)$ is, for example, the Jensen–Shannon divergence or the Wasserstein distance.
- **Variational inference.** Let $p(\theta|x)$ be a posterior distribution over parameters θ given data x. This is usually difficult to compute, so instead, we seek a good approximation $q(\theta)$. In this case, we typically let
$$J_{\text{VI}}(q) = D_{\text{KL}}(q(\theta)\,||\,p(\theta|x)),$$
where $D_{\text{KL}}(\cdot||\cdot)$ is the Kullback–Liebler divergence.
- **Reinforcement learning.** Consider a Markov decision process (S_t, A_t, R_t) governed by transitions $p(s', r|s, a)$ and a policy $\pi(a|s)$. We seek a policy π that maximizes the total discounted expected reward. In this case, we let
$$J_{\text{RL}}(\pi) = -\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R_t\Big].$$

Linearizing the probability functional

To minimize J, we need some notion of gradients of probability functionals. The appropriate generalization for probability functionals is the *von Mises influence function*.

The **von Mises influence function** of $J : \mathcal{P}(X) \to \mathbb{R}$ at $\mu \in \mathcal{P}(X)$ is a function $\Psi : X \to \mathbb{R}$ such that for all $\nu \in \mathcal{P}(X)$,
$$\mathbb{E}_{x\sim\nu}[\Psi(x)] - \mathbb{E}_{x\sim\mu}[\Psi(x)] = \lim_{\epsilon\to 0} \frac{J((1-\epsilon)\mu + \epsilon\nu) - J(\mu)}{\epsilon}.$$
It is a representation of the **Gâteaux differential**.

This construction allows for a **von Mises representation** of J, an analogue of a first-order Taylor expansion around μ_0:
$$\begin{aligned} J(\mu) &\approx J(\mu_0) + \mathbb{E}_{x\sim\mu}[\Psi(x)] - \mathbb{E}_{x\sim\mu_0}[\Psi(x)] \\ &= \mathbb{E}_{x\sim\mu}[\Psi(x)] + \text{constant}. \end{aligned}$$

Therefore, perturbing μ to decrease $\mathbb{E}_{x\sim\mu}[\Psi(x)]$ will also decrease $J(\mu)$ so long as the perturbation is small enough.

Intuitively, $\Psi : X \to \mathbb{R}$ behaves as a potential function that indicates where samples $x \sim \mu$ should descend if the goal is to decrease $J(\mu)$.

Probability functional descent

We introduce **probability functional descent** (PFD), a recipe for minimizing probability functionals $J : \mathcal{P}(X) \to \mathbb{R}$. It's a generalization of gradient descent (which is limited to functions $\mathbb{R}^n \to \mathbb{R}$).

Given an initial distribution μ, PFD updates it by repeatedly performing two steps:

- **Differentiation.** Compute the von Mises influence function of J at the current iterate μ. The influence function is a function $\Psi : X \to \mathbb{R}$.
- **Descent.** Find a distribution μ that decreases $\mathbb{E}_{x\sim\mu}[\Psi(x)]$, and set it to be the next iterate.

The descent step in practice

For the descent step, we can introduce a parameterization $\theta \mapsto \mu_\theta$ and apply gradient steps to decrease
$$\theta \mapsto \mathbb{E}_{x\sim\mu_\theta}[\Psi(x)].$$
Indeed, a generalization of the chain rule says
$$\nabla_\theta J(\mu_\theta) = \nabla_\theta \mathbb{E}_{x\sim\mu_\theta}[\Psi(x)].$$
The influence function Ψ converts a difficult minimization problem into a problem solvable by our deep learning toolbox: neural networks, stochastic gradient descent, and the reparameterization/log-derivative trick!

The differentiation step in practice

For the differentiation step, it's usually straightforward to derive an analytic expression for the influence function Ψ, from J. However, evaluating Ψ may require us to approximate it. Each approximation scheme corresponds to a variant of PFD:

- **Exact.** In some cases, it's possible to evaluate Ψ exactly, so no approximation is necessary.
- **Ad hoc.** We can look at the analytic expression for Ψ and develop ad hoc methods for approximating the terms it contains.
- **Convex duality.** If J is convex, then a generic approximation scheme is available. In this case, Ψ satisfies
$$\Psi = \underset{\varphi\in\mathcal{C}(X)}{\arg\max} \Big[\mathbb{E}_{x\sim\mu}[\varphi(x)] - J^\star(\varphi)\Big],$$
where $J^\star$ is the convex conjugate of J, and $\mathcal{C}(X)$ is the set of continuous functions on X.

  Taking advantage of this, we can model the influence function with a neural network $\varphi : X \to \mathbb{R}$ by training it to maximize $\mathbb{E}_{x\sim\mu}[\varphi(x)] - J^\star(\varphi)$.

  With this approximation scheme, PFD can be written as
$$\inf_{\mu} \sup_{\varphi} \Big[\mathbb{E}_{x\sim\mu}[\varphi(x)] - J^\star(\varphi)\Big],$$
a generalization of adversarial training!

Generative adversarial networks

When applied to J_{GM}, PFD recovers the GAN scheme, in which the influence function Ψ_{GM} is approximated by the discriminator. The differentiation step is the discriminator update; the descent step is the generator update. Different GANs use different approximation schemes:

Algorithm	Approximation scheme
Minimax GAN	Convex duality
Non-saturating GAN	Ad hoc (binary classification)
Wasserstein GAN	Convex duality

Variational inference

The influence function for J_{VI} is the ELBO integrand:
$$\Psi_{\text{VI}}(\theta) = \log\Big(\frac{q(\theta)}{p(\theta, x)}\Big).$$
PFD with different approximation schemes recovers different algorithms:

Algorithm	Approximation scheme
Black-box variational inference	Exact
Adversarial variational Bayes	Ad hoc (binary classification)
Adversarial posterior distillation	Convex duality

Actor-critic algorithms

The influence function for J_{RL} is the advantage function:
$$\Psi_{\text{RL}}(s, a) = -\frac{\sum_{t=0}^{\infty} \gamma^t p_t^\pi(s)}{\pi(s)} (Q^\pi(s,a) - V^\pi(s)).$$
The differentiation step is policy evaluation; the descent step is policy improvement. PFD with different approximation schemes recovers different algorithms:

Algorithm	Approximation scheme
Policy iteration	Exact
Policy gradient	Ad hoc (Monte Carlo)
Actor-critic	Ad hoc (least squares)
Dual actor-critic	Convex duality

Why is PFD important?

Probability functional descent allows for:

- **New algorithms.** Given a probability functional describing a new problem of interest, PFD immediately provides a recipe to minimize it.
- **Clearer understanding.** PFD clarifies relationships between existing algorithms. For example, GANs and actor-critic algorithms look similar because they both approximate the influence function, with the discriminator and critic respectively.
- **Transfer of knowledge.** PFD inspires connections that allows one field to leverage techniques from another. For example, what would happen if GAN techniques like gradient penalties were applied to value functions in RL?