Tuning Free PnP 阅读报告

赖泽强

2021 年 6 月 6 日

目录

1 基本思路

论文：Tuning-free Plug-and-Play Proximal Algorithm for Inverse Imaging Problems

在 PnP 算法，我们迭代的过程中会有一些超参数需要手工设置，比如 PnP-ADMM 中的 σ 和 ρ。不同的任务，甚至不同的图像，最优的参数是不同的。现有的实践是手工调参，但这非常费时费力。

$$\begin{aligned}
&\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}} \quad f(\boldsymbol{x}) + \frac{\rho}{2}\left\|\boldsymbol{x} - \tilde{\boldsymbol{x}}^{(k)}\right\|^2, \quad \tilde{\boldsymbol{x}}^{(k)} \equiv \boldsymbol{v}^{(k)} - \tilde{\boldsymbol{u}}^{(k)} \\
&\boldsymbol{v}^{(k+1)} = \mathcal{D}_\sigma\left(\tilde{\boldsymbol{v}}^{(k)}\right), \quad \tilde{\boldsymbol{v}}^{(k)} \equiv \boldsymbol{x}^{(k+1)} + \bar{\boldsymbol{u}}^{(k)} \\
&\bar{\boldsymbol{u}}^{(k+1)} = \bar{\boldsymbol{u}}^{(k)} + \left(\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)}\right), \quad \bar{\boldsymbol{u}}^{(k)} \equiv (1/\rho)\boldsymbol{u}^{(k)}
\end{aligned} \tag{1}$$

这篇文章用强化学习的思路去学习一个策略网络，这个策略网络的输入是迭代过程中某一步的中间变量 (比如 ADMM 中的 x, u, v, 当前的 $sigma$ 等)，输出是下一步的超参数，以及是否停止迭代。

2 详细解读

我们的目标是学习一个策略网络 $\pi_\theta(A|S)$：

- S 是输入，也是当前的状态（State），即本次迭代的各种中间变量。

- A 是输出，也是要执行的动作（Action），包括一个随机（stochastic）变量，即是否停止迭代，和两个 deterministic 的连续变量，即超参数 σ 和 ρ。

我们定义每个动作状态对的奖励为 PSNR 的增益，其中 η 是一个最小 PSNR 增益阈值，当 PSNR 增益小于这个值时，奖励为负，这么计算奖励可以鼓励网络及时停止迭代。

$$r\left(s_t, a_t\right)=\zeta\left(p\left(s_t, a_t\right)\right)-\zeta\left(s_t\right)-\eta \tag{2}$$

首先定义 return，即从当前时刻开始到一回合结束的所有奖励的总和。这里用的是 discounted return，即更近的奖励权重比较大，而远处的奖励权重比较小。

$$r_t^\gamma=\sum_{t'=0}^{N-t} \gamma^{t'} r\left(s_{t+t'}, a_{t+t'}\right) \tag{3}$$

我们的优化目标函数有多种选择，最为直接的就是最大化回报的期望，即我们希望这个策略在各种情况（trajectory）平均表现最好。

$$\theta^{\star}=\arg \max_\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{4}$$

2.1 Policy Gradient

我们使用可以梯度上升进行优化策略网络的参数，其中 $\nabla_\theta J\left(\pi_\theta\right)$ 被称为 policy gradient。

$$\theta_{k+1}=\theta_k+\left.\alpha \nabla_\theta J\left(\pi_\theta\right)\right|_{\theta_k} \tag{5}$$

推导 policy gradient：

$$\begin{aligned}
\nabla_\theta J\left(\pi_\theta\right) & =\nabla_{\theta_{\tau \sim \pi_\theta}} \mathrm{E}[R(\tau)] \\
& =\nabla_\theta \int_\tau P(\tau \mid \theta) R(\tau) \\
& =\int_\tau \nabla_\theta P(\tau \mid \theta) R(\tau) \\
& =\int_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau) \\
& =\underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\nabla_\theta \log P(\tau \mid \theta) R(\tau)\right] \\
\therefore \nabla_\theta J\left(\pi_\theta\right) & =\underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^T \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) R(\tau)\right]
\end{aligned} \tag{6}$$

事实上，$\mathrm{E}[R(\tau)]$ 这个期望是可以用蒙特卡洛法近似出来的：

$$J(\theta)=E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \approx \frac{1}{N} \sum_i \sum_t r\left(\mathbf{s}_{i, t}, \mathbf{a}_{i, t}\right) \tag{7}$$

但是这么近似出来的表达式没法对 θ 求导，所以得用上面的推导。

上面的导数是一个期望，这时候我们可以用蒙特卡洛法对导数做个近似：

$$\hat{g}=\frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) R(\tau) \tag{8}$$

其中 $|\mathcal{D}|$ 是取样的 trajectories 的数目。

在实际应用中，公式8所示的 policy gradient 存在两个很严重的问题，high variance 和 slow convergence。造成 high variance 的原因包括以下两点：

1. trajectory 的取样过程，因为我们的导数实际上一个期望，但是我们通常没法 sample 出所有情况，我们只能 sample 有限的样本，这样的话，我们的导数就很依赖样本的质量，比如，一个好的 samples 是有些 sample 的 reward 是正的，有些是负的（bad trajectory），这样求导的时候就会让正的 trajectory 概率更大，负的概率更小，但是如果 sample 出来的 bad trajectory 的 reward 也是正的，那么导数的方形就会是 bad trajectory 和 good trajectory 的平均，这样就没有之前那么好了。更糟糕的情况是，如果 sample 出来的 good trajectory 的 reward 是 0，那么导数就是 0。也就是说，导数的质量实际上取决于 sample 的 reward 的情况，而 reward 的变化是很大的，那么导数的 variance 也就很大。

2. 现有的 policy gradient 可以看出实际上一个加权的最大似然估计，我们最大化每个 action state 对的概率，与此同时每个对都被乘上了一个 reward 权重，但是现有版本乘的权重都是一样的，这样难以关照到不同 action 的重要性。

2.2 Baseline

对于第一点，从分析中我们可以知道，造成这种 high variance 的原因是 reward 不够 center，bad example 的 reward 应该全是负的，而 good example 则要全是正的。一个简单的解决方案或许是合理设计 reward function，但这么做会限制 reward function 的选择空间（没找到相关资料讲这个的）。但更常见的是将 reward 减掉一个 **baseline**。比如说下面这样：

$$\begin{aligned}&\nabla_\theta J(\theta) \approx \tfrac{1}{N} \textstyle\sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau)[r(\tau) - b] \\ &b = \tfrac{1}{N} \textstyle\sum_{i=1}^{N} r(\tau)\end{aligned} \tag{9}$$

我们用 reward 的均值来讲 reward 的 scale 尽可能变成 zero-centered。我们能这么做的原因是上述公式将 reward 减掉一个常数后，仍然是对 policy gradient 的一个无偏估计。

简单证明如下：

$$E\left[\nabla_\theta \log p_\theta(\tau) b\right] = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b d\tau = \int \nabla_\theta p_\theta(\tau) b d\tau = b \nabla_\theta \int p_\theta(\tau) d\tau = b \nabla_\theta 1 = 0 \tag{10}$$

上述推导用到了这个等式：

$$p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau) \tag{11}$$

减掉 reward 的均值是一个比较方便且还不错的 baseline，但是它并不是最优的 baseline[1]。

2.3 Reward to Go

针对第二个问题，我们的改进策略是利用 reward 的因果性，不再盲目的累加所有的 reward，而是让每一个 reward 只对当前以及历史的 action 的 gradient 进行 weighting。即将下列公式

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t} \right) \right) \left(\sum_{t=1}^{T} r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t} \right) \right) \tag{12}$$

[1]最优解推导:`https://youtu.be/VgdSubQN35g?list=PL_iWQOsE6TfURIIhCrlt-wj9ByIVpbfGc&t=816`

变成这个公式：

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t'=t}^T r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right) \tag{13}$$

其中 $\sum_{t'=t}^T r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)$ 也被成为 reward to go。

2.4 Reinforce Algorithm

上述的推导实际上构成了 reinforce algorithm，这个算法本质上就是使用策略网络 sample 出来的 trajectory 去估计 expected return，然后求出 policy gradient，优化策略网络，使用新的策略网络继续 sample，不断迭代。算法流程如下：

1. sample $\left\{\tau^i\right\}$ from $\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t'=t}^T r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right)$
3. $\theta \leftarrow \theta+\alpha \nabla_\theta J(\theta)$

因为每次都是用最新的策略网络 sample，因此 reinforce algorithm 是 on policy 的，这种方法效率实际上比较低。

2.5 Actor-Critic Algorithms

回忆在之前的 reinforce algorithm 中，我们的 policy gradient 长这样：

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t'=t}^T r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right) \tag{14}$$

我们将 $\hat{Q}_{i,t}^\pi$ 称为 reward to go。

$$\hat{Q}_{i,t}^\pi=\sum_{t'=t}^T r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right) \tag{15}$$

事实上，$\hat{Q}_{i,t}^\pi$ 实际上是对我们在 $s_{i,t}$ 这个状态下使用 $a_{i,t}$ 这个 action 的奖励的一个**估计**，我们希望通过这个奖励让策略网络学会如何选择正确的 action（即奖励大的 action 概率大）。$\hat{Q}_{i,t}^\pi$ 是对真实 reward to go 的一个蒙特卡洛估计（样本数为 1，可以想象，这会造成 high variance），而真实的 reward to go 则是下面这个表达式，每一步奖励都是对从 t 时刻开始 trajectory 的一个期望（我们不知道策略网络选择什么样的路径，但我们希望期望的 reward 最大，即“平均 reward”最大）。

$$Q\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{t'=t}^T E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t, \mathbf{a}_t\right] \tag{16}$$

因此，更好的 policy gradient 实际上应该用 reward to go 的 expection：

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right) Q\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \tag{17}$$

然后，使用 Baseline 可以进一步降低 variance，并且在这种情况下，使用 $V(s_t)$ 会比较平均 reward 更好。

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(Q\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)-V\left(\mathbf{s}_{i,t}\right)\right) \tag{18}$$

$$V\left(\mathbf{s}_t\right)=E_{\mathbf{a}_t \sim \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left[Q\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{19}$$

下面总结出现过得几个重要函数：

- State Action Function：状态动作函数是在当前策略下，在 s_t 采取 a_t 的 total reward 的期望。

$$Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{t'=t}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t, \mathbf{a}_t\right]$$

- Value Function: 价值函数是在当前策略下，状态 s_t 的 total reward 的期望，即把 s_t 采取的动作也做个平均。

$$V^{\pi}\left(\mathbf{s}_t\right)=E_{\mathbf{a}_t \sim \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left[Q\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$$

- Advantage Function: 优势函数是前两者的差，意义是 a_t 相对于平均情况的好坏程度。

$$A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right)=Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right)-V^{\pi}\left(\mathbf{s}_t\right)$$

根据 V 的定义，我们之前的优化目标（最大化 total reward 的期望）实际上可以写成下面的形式:

$$J(\theta)=E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]=E_{\mathbf{s}_1 \sim p\left(\mathbf{s}_1\right)}\left[V^{\pi}\left(\mathbf{s}_1\right)\right] \tag{20}$$

大部分论文，博客会用前者推导 policy gradient，但有些会用 V 的形式推导，但二者实际上是一样的。

现在的问题是，不管是 Q，V 还是 A，它们都是期望，我们没法直接算它们，如果使用蒙特卡洛估计，那么我们将退回原来的 reinforce 算法。我们需要使用 function approximator 去估计这些期望（比如神经网络）。

我们有多种选择，我们可以同时使用两个 function approximator 分别估计 Q，V，也可以用一个 function approximator 直接估计 A。区别在于，使用多个 function approximator 可能不太好优化，直接估计 A 则训练算法比较难设计（具体看后面）。

在现有实践中，我们一般选择估计 V，因为 V 通常更好估计（V 只依赖于 state），不依赖于 action，而且 Q 可以由 V 导出。

推导如下：

$$\begin{aligned} Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) &=r\left(\mathbf{s}_t, \mathbf{a}_t\right)+\sum_{t'=t+1}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t, \mathbf{a}_t\right] \\ &=r\left(\mathbf{s}_t, \mathbf{a}_t\right)+E_{\mathbf{s}_{t+1} \sim p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)}\left[V^{\pi}\left(\mathbf{s}_{t+1}\right)\right] \\ &\approx r\left(\mathbf{s}_t, \mathbf{a}_t\right)+V^{\pi}\left(\mathbf{s}_{t+1}\right) \end{aligned} \tag{21}$$

第一个等号，s_t, a_t 的 reward 是一个定值，可以从期望里提出来。第二个等号，value function 是对 action 的期望，如果我们再对 state 做一个期望，那它就是 Q 了（Q 是对 state 和 action 二者的期望）。第三个等号，单样本蒙特卡洛估计。

2.5.1 Approximate Value Function

我们使用 Supervised training 的方式训练 Value Function Approximator。首先通过某种方式收集一些数据 $\{(s_t, y_t)\}$，然后使用下面的 Loss 进行训练。

$$
\mathcal{L}(\phi)=\frac{1}{2}\sum_{i}\left\|\hat{V}_{\phi}^{\pi}\left(\mathbf{s}_{i}\right)-y_{i}\right\|^{2} \tag{22}
$$

训练数据的 s_t 很好办，直接让 agent 根据策略网络 play 一下就有了，主要是 gt 的 y_t 要怎么得到。一种最直接的方式是用单样本蒙特卡洛估计：

$$
y_{t}=\sum_{t'=t}^{T} r\left(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'}\right) \tag{23}
$$

但更好的方式使用 TD 算法：

$$
y_{i,t}=\sum_{t'=t}^{T} E_{\pi_{\theta}}\left[r\left(\mathbf{s}_{t'},\mathbf{a}_{t'}\right) \mid \mathbf{s}_{i,t}\right] \approx r\left(\mathbf{s}_{i,t},\mathbf{a}_{i,t}\right)+V^{\pi}\left(\mathbf{s}_{i,t+1}\right) \approx r\left(\mathbf{s}_{i,t},\mathbf{a}_{i,t}\right)+\hat{V}_{\phi}^{\pi}\left(\mathbf{s}_{i,t+1}\right) \tag{24}
$$

第一个约等于是蒙特卡洛估计，第二个约等于用已有的网络估计 $V^{\pi}\left(\mathbf{s}_{i,t+1}\right)$，但是因为 $r\left(\mathbf{s}_{i,t},\mathbf{a}_{i,t}\right)$ 是真实的，所以这个估计是对 $y_{i,t}$ 更真实的估计。这个方法在一些地方也被称为 **bootstrap**。

2.5.2 Summary

图**??**展示了 actor-critic 算法的流程，其中增加了一个 discount factor 优化。值得注意的是 Actor-Critic 仍然是一个 on policy 的算法，但是因为只需要采样一个动作，不需要完成的 trajectory，所以 actor-critic 还是比 reinforce 效率高的。通常做法是，先运行一段时间，获得一个 batch 的 samples，然后进行几步优化；但也可以采用 online 的方法，没采样的动作，就优化一次。为了降低各个 sample 的相关性，一些方法会采用多个 worker 来收集 sample，这也 worker 之间是不关联的，不过 worker 的连续 step 仍然是关联的。

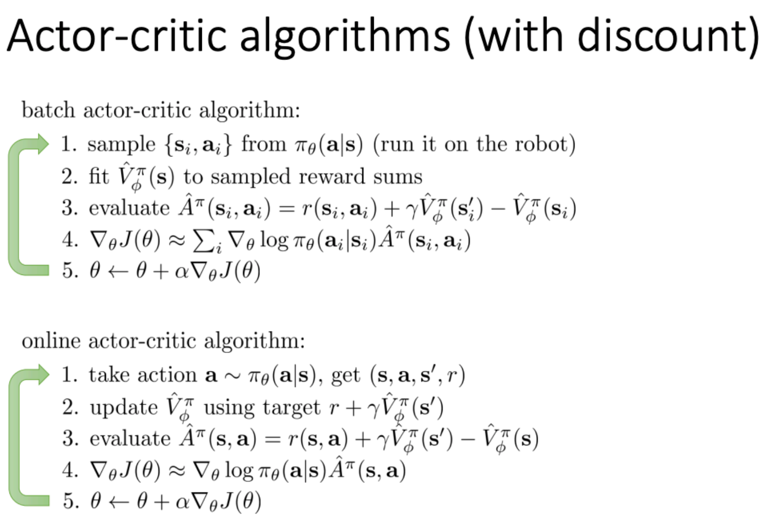

图 1: Actor-critic policy gradient 算法流程

2.6 Deterministic Policy Gradient

前面几章讲的都是针对离散控制的策略，在 TF-PnP 中，离散控制只有一个 terminal time（是否停止迭代），剩下的参数都是连续变量。

回忆一下离散控制的 policy gradient：

$$
\begin{aligned}
\nabla_\theta J(\theta) &\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t'=t}^{T} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right) \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right) Q\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)
\end{aligned} \tag{25}
$$

确定性的策略网络只会输出一个 action，因此原来离散控制中的 log probability 就没有了。

$$
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta Q\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \tag{26}
$$

在实际训练的时候，我们可以使用 TD 算法优化公式26。

2.7 Target Network

使用目标网络可以减轻 bootstrap 带来的偏差。

如何理解 bootstrap 的偏差？考虑下面这个 bootstrap 的公式：

$$
V^\pi\left(\mathbf{s}_{i,t}\right) \approx r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) + \hat{V}_\phi^\pi\left(\mathbf{s}_{i,t+1}\right) \tag{27}
$$

如果我们每次都用上一步的价值网络预测 V_{t+1}，那么 V_{t+1} 的误差会在迭代过程中不断传播，即当我们下一次优化的时候，新的价值网络误差点更多了，迭代一次又会产生一个误差点更更多的价值网络。

使用一个距离比较远，且参数固定的目标网络估计 V_{t+1} 能一定程度上解决这个问题，因为目标网络是固定的，每次迭代误差点就那么多，不会造成累加。（当然这没有完全解决 bootstrap 的偏差问题）。

随着迭代进行，间隔一段时间同步一下目标网络（通常使用加权更新参数的方式同步）。

$$
\boldsymbol{w}_{\text{new}}^{-} \leftarrow \tau \cdot \boldsymbol{w}_{\text{new}} + (1-\tau) \cdot \boldsymbol{w}_{\text{now}}^{-} \tag{28}
$$