The policy search problem

We want to optimize $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$, where we tune the policy parameters θ, thus the goal it so find,

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau|\theta)R(\tau)$$

where $P(\tau|\theta)$ is the probability of trajectory τ under policy π_θ. This direct policy search is black box optimization, in short a function $f(x) : \mathbb{R}^n \to \mathbb{R}$ for which the analytic form is not know. One way of finding a minimum is by approximating the gradient for $f(x)$. In policy search this can be done by the *Policy Gradient Theorem*.

The general idea is to increase $P(\tau|\theta)$ for trajectories τ with a high return. Doing so by following gradient ascent from the analytical knowledge that is gathered. Now as we have access to the state, action and reward at each step, we're not applying black-box optimization anymore. However, the transition and reward functions are still unknown (gray-box).

Policy Gradient Theorem
Hence by looking for parameters $\theta^* = \arg\max_\theta J(\theta)$ we look for the gradient,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \sum_\tau P(\tau \mid \boldsymbol{\theta}) R(\tau) \\
&\text{The gradient of sum, is the sum of the gradients} \\
&= \sum_\tau \nabla_{\boldsymbol{\theta}} P(\tau \mid \boldsymbol{\theta}) R(\tau) \\
&\text{Multiply by one} \\
&= \sum_\tau \frac{P(\tau \mid \boldsymbol{\theta})}{P(\tau \mid \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} P(\tau \mid \boldsymbol{\theta}) R(\tau) \\
&= \sum_\tau P(\tau \mid \boldsymbol{\theta}) \frac{\nabla_{\boldsymbol{\theta}} P(\tau \mid \boldsymbol{\theta})}{P(\tau \mid \boldsymbol{\theta})} R(\tau) \\
&\text{The property of the gradient of log} \\
&= \sum_\tau P(\tau \mid \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau \mid \boldsymbol{\theta}) R(\tau) \\
&\text{By the definition of the expectation} \\
&= \mathbb{E}_\tau [\nabla_{\boldsymbol{\theta}} \log P(\tau \mid \boldsymbol{\theta}) R(\tau)]
\end{aligned}$$

We want to compute the expectation but don't have the analytical expression for $P(\tau|\theta)$. Thus, we can't compute the gradient term. Now, we look at reformulating $P(\tau|\theta)$ using the policy π_θ. The probability of a trajectory by the policy for a horizon H can be written as,

$$P(\tau|\theta) = \Pi_{t=1}^H p(s_{t+1}|s_t, a_t) \cdot \pi_\theta(a_t|s_t)$$

Note, that the markov assumption is used here, this statement hoplds if the steps are independent. Hence, under the Markov assumption,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log \mathrm{P}(\tau, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \log \left[\prod_{t=1}^{H} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \cdot \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)\right] \\
&\text{log of product is sum of logs} \\
&= \nabla_{\boldsymbol{\theta}} \left[\sum_{t=1}^{H} \log \mathrm{p}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) + \sum_{t=1}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)\right] \\
&\text{As the first term is independent of } \boldsymbol{\theta} \\
&= \nabla_{\boldsymbol{\theta}} \sum_{t=1}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)^* \\
&\text{Again, we can switch out the gradient and sum} \\
&= \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)^*
\end{aligned}$$

Now, the key insight is that we know $\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$. Filling in this result we obtain,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) R(\tau)\right]$$

Naive Policy Gradient Algorithm

This can be used for direct policy search on $J(\theta)$ to gradient ascent on π_θ. The expectation can be approximated by sampling over m trajectories,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right) R\left(\tau^{(i)}\right)$$

This algorithm would follow the following steps,

1. Sample a set of trajectories from π_θ.
2. Compute, $Loss(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right) R\left(\tau^{(i)}\right)$
3. Minimize the loss by back-propagating the neural network using a optimizer

Limitations

The limitations of this algorithm are, that we will need a large batch of trajectories or the algorithm will suffer from large variance. The sum of rewards is not much informative, when $R(\tau) = 0$, the value will not change. Finally, computing R from complete trajectories is not the best thing we can do. Hence in next section we consider a baseline.

Considering a baseline

We consider the gradient term of our objective function again,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right) R\left(\tau^{(i)}\right) \\
&\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right)\left[\sum_{t=1}^{H} r\left(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}\right)\right] \\
&\text{We split up the reward} \\
&\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right)\left[\sum_{k=1}^{t-1} r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right)+\sum_{k=t}^{H} r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right)\right] \\
&\text{Pas rewards do not depend on the current action} \\
&\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right)\left[\sum_{k=t}^{H} r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right)\right] \\
&\text{Reduce the variance by discounting the rewards along the trajectory} \\
&\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right)\left[\sum_{k=t}^{H} \gamma^{k-t} r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right)\right]
\end{aligned}$$

We have essentially rewritten our loss term of the algorithm, we slightly reduced the variance by discounting rewards along the trajectory. We can rewrite the discounted reward of the trajectory with the Q-function, this gives us,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right) Q_{(i)}^{\pi_\theta}\left(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}\right)$$

This suggests that the gradient could be a function of the local step, if $Q_{(i)}^{\pi_\theta}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$ is estimated in one step. To estimate $Q_{(i)}^{\pi_\theta}$ we build a model $\hat{Q}_\phi^{\pi_\theta}$ through function approximation. To update this model there are two approaches, namely, with a Monte Carlo estimate or Temporal Difference estimate.

Monte Carlo estimate of Q
First, by utilizing Monte Carlo we will be looking at the expectation of Q, $Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{(i)}[Q_{(i)}^{\pi_\theta}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})]$. This estimate will result in regression against the empirical return,

$$\phi_{j+1} \rightarrow \arg\min_{\phi_j} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H}\left(\sum_{k=t}^{H} \gamma^{k-t} r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right)-\hat{Q}_{\phi_j}^{\pi\theta}\left(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}\right)\right)^2$$

Temporal Difference estimate of Q
Second, the usage of Temporal Difference estimate requires us to initialize $\hat{Q}_{\phi_0}^{\pi_\theta}$ and fit using $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ data,

$$\begin{aligned}
&\phi_{j+1} \rightarrow \min_{\phi_j} \sum_{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')}\left\|r+\gamma f\left(\hat{Q}_{\phi_j}^{\pi_\theta}(\mathbf{s}', .)\right)-\hat{Q}_{\phi_j}^{\pi_\theta}(\mathbf{s}, \mathbf{a})\right\|^2 \\
&\text{Different methods could apply different approaches to this,} \\
&f\left(\hat{Q}_{\phi_j}^{\pi_\theta}(\mathbf{s}', .)\right)=\max_{\mathbf{a}} \hat{Q}_{\phi_j}^{\pi_\theta}(\mathbf{s}', \mathbf{a})(\text{ Q-learning }),=\hat{Q}_{\phi_j}^{\pi_\theta}(\mathbf{s}', \pi_\theta(\mathbf{s}'))(\mathrm{AC}) \ldots
\end{aligned}$$

In both of these cases it might be necessary to utilize regularization to prevent large steps in ϕ.

Finally, we can consider using a *baseline* b in our result, which was originally introduced by Williams, 1992. The paper in which the REINFORCE algorithm is introduced.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right) \left[\sum_{k=t}^{H} \gamma^k r\left(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}\right) - b\right]$$

We can pick several different values for $b/b(s_t^{(i)})$, a first baseline could be taking the average return $\bar{r}$ over all states of the batch. In which the probability would go down if the return is worse than average. Furthermore, taking $(r_t^{(i)} - \bar{r})/\sigma_r$ is a renormalization of the value and hence improves the variance. However, keep in mind this does not work if all rewards are identical, e.g. the CartPole environment.

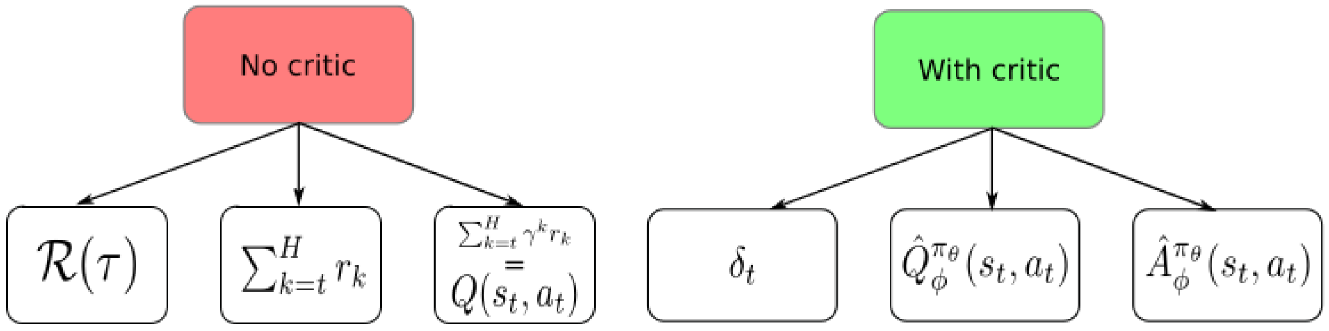
It should be kept in mind that any choice of b suffices, as long at it does not depend on the action, otherwise this would influence the gradient. Now, better picks for the baseline are,

- State-dependent expected return:
  $b(\mathbf{s}_t) = V^\pi(\mathbf{s}_t) = \mathbb{E}_\tau\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{H-t} r_H\right]$
  This baseline, increases the log probability of the action proportionally to how much its returns are better than the expected return under the current policy.

- Advantage function: $A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t \mid \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$

Again the estimation is similar to that of the Q-function, but simpler again we can pick between Monte-Carlo and Temporal Difference estimates.

State-dependent baseline
Finally there are several possibilities for picking a baseline, namely,
$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \pi_\theta(.)}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}\right)\right] \psi_t \text{ where } \psi_t \text{ can be:}$



No critic
With critic
$\mathcal{R}(\tau)$
$\sum_{k=t}^{H} r_k$
$\sum_{k=t}^{H} \gamma^k r_k = Q(s_t, a_t)$
δ_t
$\hat{Q}_\phi^{\pi_\theta}(s_t, a_t)$
$\hat{A}_\phi^{\pi_\theta}(s_t, a_t)$

- $\sum_{t=0}^{H} \gamma^t r_t$: total (discounted) reward of trajectory
- $\sum_{k=t}^{H} \gamma^{k-t} r_k$: sum of rewards after $\mathbf{a}_t$

- $\sum_{k=t}^{H} \gamma^{k-t} r_k - b(\mathbf{s}_t)$: sum of rewards after $\mathbf{a}_t$ with baseline
- $\delta_t = r_t + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$: error, with $V^\pi(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t}\left[\sum_{k=0}^{H} \gamma^k r_{t+l}\right]$
- $\hat{Q}_\phi^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{a_{t+1}}\left[\sum_{k=0}^{H} \gamma^k r_{t+l}\right]$: action-value function
- $\hat{A}_\phi^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) = \hat{Q}_\phi^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_t) = \mathbb{E}[\delta_t]$, advantage function

The **Vanilla Policy Gradient** algorithm would be executing as follows,

---

Vanilla Policy Gradient

```
Initialize policy parameter $\theta$, baseline $b$
**for** iteration$= 1, 2, \ldots$ **do**
    Collect a set of trajectories by executing the current policy
    At each timestep in each trajectory compute
     the return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
     the *advantage estimate* $\hat{A} = R_t - b(s_t)$.
    Re-fit the baseline, by minimizing $||b(s_t) - R_t||^2$,
     summed over all trajectories and timesteps.
    Update the policy, using a policy gradient estimate $\hat{g}$,
     which is a sum of all terms $\nabla \log \pi(a_t|s_t, \theta)\hat{A}_t$
**end for**
```

- TODO: Review High-dimensional continuous control using generalized advantage estimation paper here, and useful notes on this

References

- The deepRL bootcamp lectures by Peter Abbeel
- The RLVS lectures by Olivier Sigaud
- The blog post and lecture by Karpathy
- REINFORCE paper by Williams
- High-Dimensional Continuous Control Using Generalized Advantage Estimation