Markov Decision Processes Lecture Notes 09

Policy Gradient Methods

Authors: Benjamin Van Roy *November 17, 2025*

`Reinforce` is a simple algorithm that adjusts a policy to increase probabilities assigned to actions that result in desirable outcomes and decrease probabilities of actions that result in undesirable outcomes. We will begin by studying the algorithm in the simple context of multi-armed bandits.

1 Multi-Armed Bandits

Consider a decision problem where we must choose among a finite set $\mathcal{A}$ of actions. And upon choice of an action A_0, we observe a random reward R_1. Then, we choose A_1 and observe a reward R_2. And so on. We assume that, for each action $a \in \mathcal{A}$, the distribution over rewards that results from that action is fixed across time.

Such a decision problem is often referred to as a bandit. The term *bandit* is a somewhat antiquated reference to slot machines, which "steal" money from the player. Think of each action $a \in \mathcal{A}$ as an arm of a slot machine. Each time an arm A_t is pulled, a reward R_{t+1} is generated. Over many turns, a player may learn which arm to pull to generate the largest reward on average.

1.1 Reinforce

The algorithm we consider maintains a parameterized policy π_θ, which assigns a probability $\pi_\theta(a)$ to each action $a \in \mathcal{A}$. We denote the parameter vector at time t by θ_t. Action A_t is sampled from π_{θ_t}. Upon observation of the resulting reward R_{t+1}, the parameters are updated according to

$$\theta_{t+1} = \theta_t + \alpha_t R_{t+1} \nabla_\theta \ln \pi_{\theta_t}(A_t), \tag{1}$$

where α_t is a step size. This update rule is intuitive: the larger (or smaller) the value of R_{t+1} the more it increases (or decreases) the probability $\pi_{\theta_t}(A_t)$ assigned to the action A_t. In this way, the algorithm reinforces actions with desirable outcomes. The reward R_{t+1} serves as a *reinforcement signal*.

1.2 An Example: Gibbs Distribution

To offer a simple example, consider a parameterized policy that samples actions according to a Gibbs distribution:

$$\pi_\theta(a) = \frac{e^{\theta_a}}{\sum_{a' \in \mathcal{A}} e^{\theta_{a'}}}. \tag{2}$$

Think of each parameter θ_a as a score assigned to action a. Actions with higher scores are more likely to be sampled by the policy.

The gradient of the log-probability $\pi_\theta(a)$ can be written as

$$\begin{aligned}
\nabla_\theta \ln \pi_\theta(a) =& \nabla_\theta \left(\theta_a - \ln \sum_{a' \in \mathcal{A}} e^{\theta_{a'}} \right) && (3)\\
=& \mathbf{1}_a - \frac{1}{\sum_{a'' \in \mathcal{A}} e^{\theta_{a''}}} \sum_{a' \in \mathcal{A}} e^{\theta_{a'}} \mathbf{1}_{a'} && (4)\\
=& \mathbf{1}_a - \sum_{a' \in \mathcal{A}} \pi_\theta(a') \mathbf{1}_{a'}. && (5)
\end{aligned}$$

For each action a, this is the one-hot vector, offset by a vector that depends on θ but not a such that the expectation across actions is zero. With this expression, the parameter update (1) becomes

$$\theta_{t+1,a} = \begin{cases} \theta_{t,a} + \alpha_t(1 - \pi_{\theta_t}(a))R_{t+1} & \text{if } a = A_t \\ \theta_{t,a} - \alpha_t \pi_{\theta_t}(a) R_{t+1} & \text{otherwise.} \end{cases} \tag{6}$$

Consider a concrete case where $\mathcal{A} = \{0, 1\}$, and actions generate Bernoulli rewards with success probabilities $p_0 = 0.6$ and $p_1 = 0.4$. Hence, the optimal action is 0. Figure 1 plots results from applying `reinforce` with a step size of 0.05. As iterations progress, the action probability concentrates on the optimal action. The score of the optimal action also increases over iterations.

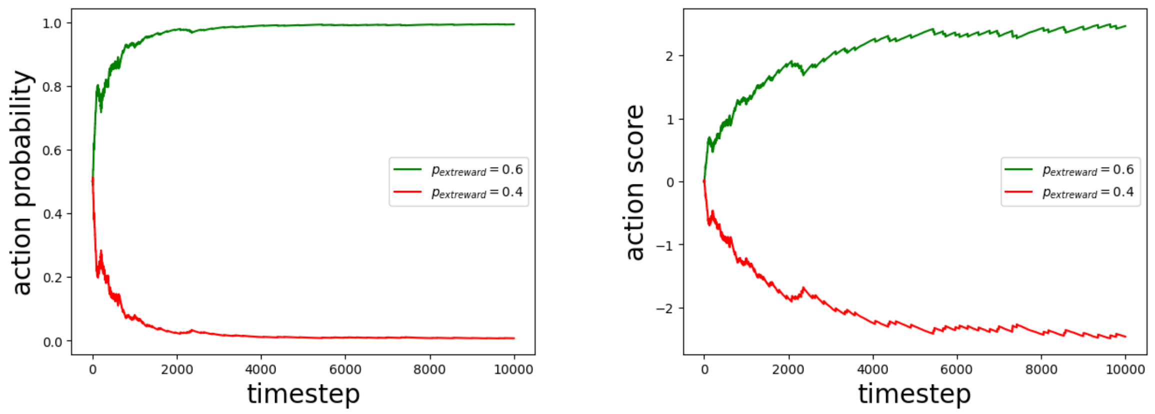

Figure 1: Probs and scores.

1.3 The Policy Gradient

Let $r(a) = \mathbb{E}[R_{t+1}|A_t = a]$ denote the average reward generated by action a. Consider a parameterized policy π_θ such that $\pi_\theta(a) > 0$ for all θ and a. The average reward attained by a parameterized policy π_θ is

$$\lambda_\theta = \sum_{a \in \mathcal{A}} \pi_\theta(a) r(a). \tag{7}$$

`Reinforce` improves the average reward by approximately following the policy gradient $\nabla_\theta \lambda_\theta$. To understand how this works, note that the policy gradient can be written as

$$\begin{aligned} \nabla_\theta \lambda_\theta =& \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(a) r(a) && (8) \\ =& \sum_{a \in \mathcal{A}} r(a) \nabla_\theta \pi_\theta(a) && (9) \\ =& \sum_{a \in \mathcal{A}} \pi_\theta(a) r(a) \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} && (10) \\ =& \sum_{a \in \mathcal{A}} \pi_\theta(a) r(a) \nabla_\theta \ln \pi_\theta(a) && (11) \end{aligned}$$

Hence,

$$\nabla_{\theta_t} \lambda_{\theta_t} = \mathbb{E}[\mathbb{E}[R_{t+1}|A_t] \nabla_{\theta_t} \ln \pi_{\theta_t}(A_t) | \theta_t] = \mathbb{E}[R_{t+1} \nabla_{\theta_t} \ln \pi_{\theta_t}(A_t) | \theta_t] \tag{12}$$

In other words, the expectation of the `reinforce` update $R_{t+1} \nabla_\theta \ln \pi_{\theta_t}(A_t)$ is the policy gradient. Scaling by a suitable step size induces the averaging needed so that the trajectory followed by the algorithm approximates a flow along the policy gradient.

1.4 Baselining

Consider a variation of the two-armed bandit discussed earlier, but with reward probabilities $p_0 = 0.96$ and $p_1 = 0.94$. The optimal action remains action 0, but both actions deliver reward 1 with high probability.

Probabilities $\pi_{\theta_t}(0)$ of choosing the optimal action as parameters are adapted by `reinforce` are plotted in Figure 2 (red). The trajectory is erratic and does not reliable converge on an optimal policy. The erratic behavior stems from chattering of parameters due to the fact that each time an action is selected and a reward of 1 is observed, its score is increased while the score of the other action is decreased. This erratic behavior can be abated if the step size is reduced to a very small value, which can lead to the desired convergence through at a very slow convergence rate.

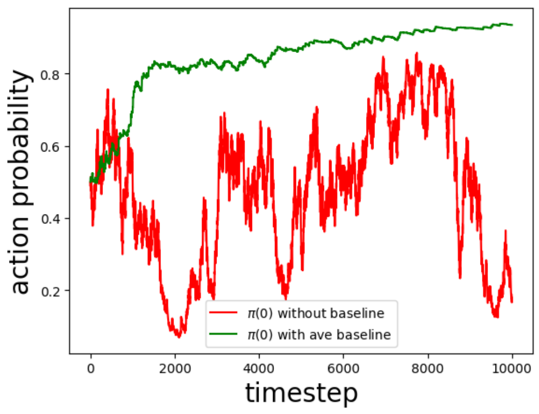

Figure 2: Without and with baseline.

One way to operate effectively with a larger step size and maintain fast and reliable convergence is to baseline rewards. Baselining involves subtracting from reward a statistic reduces their magnitudes. With baselining, the algorithm becomes

$$\theta_{t+1} = \theta_t + \alpha_t(R_{t+1} - B_t)\nabla_\theta \ln \pi_{\theta_t}(A_t), \tag{13}$$

where B_t is a baseline. Figure 2 (green) plots results where $B_t = \frac{1}{t}\sum_{k=1}^t R_k$ is the running average of rewards. This trajectory reliably converges to 1 and at a reasonable rate.

It is worth noting that a baselined reinforcement signal $R_{t+1} - B_t$ remains in expectation a policy gradient. In particular, for any B_t that is determined by θ_t and the history H_t, we have

$$\begin{aligned}
\mathbb{E}[(R_{t+1} - B_t)\nabla_{\theta_t} \ln \pi_{\theta_t}(A_t)|\theta_t, H_t] =& \mathbb{E}[R_{t+1}\nabla_{\theta_t} \ln \pi_{\theta_t}(A_t)|\theta_t] - \mathbb{E}[B_t\nabla_{\theta_t} \ln \pi_{\theta_t}(A_t)|\theta_t, H_t] && (14)\\
=& \nabla_{\theta_t}\lambda_{\theta_t} - B_t \sum_{a\in\mathcal{A}} \pi_{\theta_t}(a)\nabla_{\theta_t} \ln \pi_{\theta_t}(a) && (15)\\
=& \nabla_{\theta_t}\lambda_{\theta_t} - B_t \sum_{a\in\mathcal{A}} \nabla_{\theta_t}\pi_{\theta_t}(a) && (16)\\
=& \nabla_{\theta_t}\lambda_{\theta_t} - B_t\nabla_{\theta_t} \sum_{a\in\mathcal{A}} \pi_{\theta_t}(a) && (17)\\
=& \nabla_{\theta_t}\lambda_{\theta_t} - B_t\nabla_{\theta_t} 1 && (18)\\
=& \nabla_{\theta_t}\lambda_{\theta_t}. && (19)
\end{aligned}$$

2 Average Return

The reinforce algorithm extends naturally to optimization of average return in a unichain MDP $(\mathcal{S}, \mathcal{A}, P)$ with a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

2.1 The Policy Gradient Theorem

Consider a policy π_θ that satisfies $\pi_\theta(a|s) > 0$ for all θ, s, and a. Recall that, for each policy π, Q_π denotes the relative action value function, which solves

$$Q_\pi(s,a) = r(s,a) - \lambda_\pi + \sum_{s' \in \mathcal{S}} P_{ass'} \max_{a' \in \mathcal{A}} Q_\pi(s', a') \qquad \forall s \in \mathcal{S}, a \in \mathcal{A},$$

where λ_π is the average return

$$\lambda_\theta = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_\theta(s,a) r(s,a). \tag{20}$$

As shown by Sutton et al. [1999], Konda and Tsitsiklis [1999],

$$\nabla_\theta \lambda_\theta = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_\theta(s,a) Q_{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s). \tag{21}$$

This formula is often referred to as *the policy gradient theorem.*

2.2 Reinforce for Maximizing Gain

The policy gradient theorem (21) motivates a version of the `reinforce` algorithm that uses the action value as a reinforcement signal:

$$\theta_{t+1} = \theta_t + \alpha_t Q_{\pi_{\theta_t}}(S_t, A_t) \nabla_{\theta_t} \ln \pi_{\theta_t}(A_t|S_t). \tag{22}$$

Similarly with the multi-armed bandit, this update rule can be very slow without baselining. A natural baseline is the value $V_{\pi_\theta}(S_t)$. This gives rise to an update of the form

$$\theta_{t+1} = \theta_t + \alpha_t (Q_{\pi_{\theta_t}}(S_t, A_t) - V_{\pi_{\theta_t}}(S_t)) \nabla_\theta \ln \pi_\theta(A_t|S_t). \tag{23}$$

These algorithms are reminiscent of policy iteration. Recall that policy iteration operates by replacing an iterate π with a new policy π' that is greedy with respect to Q_π. In a similar spirit, these policy gradient algorithms in tend to increase probabilities of actions a with larger action values $Q_{\pi_{\theta_t}}(S_t, a)$ and decrease probabilities of actions with smaller action values.

2.3 Rollouts

An impediment to using either (22) or (23) as an update rule is that it requires knowledge of $Q_{\pi_{\theta_t}}(S_t, A_t)$ and/or $V_{\pi_{\theta_t}}(S_t)$. One way to address that is by simulating rollouts that can be used to generate estimates $\hat{Q}_t \approx Q_{\pi_{\theta_t}}(S_t, A_t)$ and/or $\hat{V}_t \approx V_{\pi_{\theta_t}}(S_t)$ of the value and action value.

An extreme version of this is where a single rollout is used as an estimate of $Q_{\pi_{\theta_t}}(S_t, A_t)$. In particular, simulate a trajectory $\tilde{S}_0, \tilde{A}_0, \tilde{S}_1, \tilde{A}_1, \ldots$ under policy π_{θ_t} initialized with $\tilde{S}_0 = S_0$ and $A_0 = \tilde{A}_0$ and, given the rewards $\tilde{R}_1, \tilde{R}_2, \ldots$, let

$$\hat{Q}_t = \sum_{k=0}^{\infty} (\tilde{R}_{k+1} - \tilde{\lambda}),$$

where $\tilde{\lambda} = \lim_{K \to \infty} \frac{1}{K} \sum_{k=0}^{K-1} \tilde{R}_{k+1}$. The policy parameters can then be updated according to

$$\theta_{t+1} = \theta_t + \alpha_t \hat{Q}_t \nabla_{\theta_t} \ln \pi_{\theta_t}(A_t|S_t).$$

2.4 Value Function Approximation via Temporal-Difference Learning

An alternative is to retain and update an approximation of Q_{π_θ}. In particular, we can use a parameterized policy $\tilde{Q}_\psi$ and at each time update the parameters ψ based on observed actions, state transitions, and rewards, steering the current approximation $\tilde{Q}_{\psi_t}$ toward the action value function $Q_{\pi_{\theta_t}}$ of the current policy. Then, we can apply an update of the form

$$\theta_{t+1} = \theta_t + \alpha_t \left(\tilde{Q}_{\psi_t}(S_t, A_t) - \sum_{a \in \mathcal{A}} \pi_\theta(a|S_t) \tilde{Q}_{\psi_t}(S_t, a) \right) \nabla_{\theta_t} \ln \pi_{\theta_t}(A_t|S_t). \tag{24}$$

Temporal-difference learning provides an algorithm for updating ψ_t.

$$d_{t+1} = R_{t+1} - \lambda_t + \tilde{Q}_{\psi_t}(S_{t+1}, A_{t+1}) - \tilde{Q}_{\psi_t}(S_t, A_t), \tag{25}$$

$$\psi_{t+1} = \psi_t + \beta_t d_{t+1} \nabla_{\psi_t} \tilde{Q}_{\psi_t}(S_t, A_t), \tag{26}$$

$$\lambda_{t+1} = \lambda_t + \beta_t d_{t+1}. \tag{27}$$

Here, d_{t+1} is a temporal difference, λ_t is an estimate of the gain, and β_t is a step size.

3 Total Return

The algorithmic ideas we have discussed carry over to maximization of total return over a horizon T.

3.1 Policy Gradient Algorithm

When using a policy gradient algorithm to maximize total return, it is often convenient to update parameters once per simulated episode. By *episode*, we mean a single trajectory $S_0, A_0 \ldots, S_{T-1}, A_{T-1}, S_T$. Given policy parameters θ_ℓ and such a simulated episode, we consider the update rule

$$\theta_{\ell+1} = \theta_\ell + \alpha_\ell \sum_{t=0}^{T-1} Q_{\pi_{\theta_\ell}}(S_t, A_t) \nabla_{\theta_\ell} \ln \pi_{\theta_\ell}(A_t|S_t). \tag{28}$$

As in the previous section, an obstacle to applying this update rule is that the value function is not available. Again, this can be addressed by using estimates, either based on rollouts or an incrementally updated parameterized approximations to the action value function.

Since the policy π_{θ_ℓ} remains unchanged over the ℓ episode, it is natural to use as a rollout starting at (S_t, A_t) the rewards of $R_{t+1}, \ldots, R_{T-1}$ from the same simulated trajectory. This gives rise to an estimate $\hat{Q}_t = \sum_{k=t}^{T-1} R_{k+1}$ and an update rule

$$\theta_{\ell+1} = \theta_\ell + \alpha_\ell \sum_{t=0}^{T-1} \hat{Q}_t \nabla_{\theta_\ell} \ln \pi_{\theta_\ell}(A_t|S_t). \tag{29}$$

One can also design more efficient policy gradients that approximte the baselined version of (30) :

$$\theta_{\ell+1} = \theta_\ell + \alpha_\ell \sum_{t=0}^{T-1} (Q_{\pi_{\theta_\ell}}(S_t, A_t) - V_{\pi_{\theta_\ell}}(S_t)) \nabla_{\theta_\ell} \ln \pi_{\theta_\ell}(A_t|S_t). \tag{30}$$

3.2 Application to Language Models

Versions of `reinforce` are used in training today's language models. For example, in training language models to generate satisfactory responses where they do not already do so, especially when satisfactory responses are difficult to generate but easy to verify. For example, suppose that there are challenging math problems of a particular kind that are not successfully solved by a language model. Suppose humans also find it difficult to produce correct solutions but easy to check whether a given solution is correct. Then, through repeated trial and error a policy gradient method can train the language model to generate correct solutions.

Figure 3 illustrates a prototypical large language model (LLM). A context window of N tokens is taken as input, and a predictive distribution of the next token is generated as output. Tokens are elements of a vocabulary, typically consisting of words, punctuation, or other special symbols used to inform the LLM.

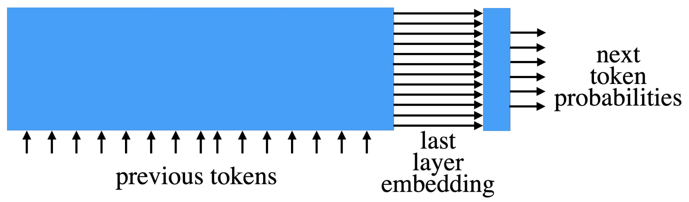

Figure 3: An autoregressive language model.

The LLM can be thought of as implementing a policy for an MDP. The state S_t is the context window and the action A_t is the next token. The LLM generates a distribution $\pi_\theta(\cdot|S_t)$ from which A_t is sampled. The parameters θ are the weights and biases in the neural network that implements the LLM. The next state S_{t+1} is produced by dequeueing the least-recent token from the context and enqueueing A_t.

`Reinforce` can be applied to train the LLM over episodes. For example, in each ℓth episode, the initial context window S_0 could express a math problem, and then π_{θ_ℓ} can be used to generate a sequence of T tokens $A_0, \ldots, A_{T-1}$. A reward signal R_T might then indicate whether the sequence provides a correct answer to the math problem.

References

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

Richard S Sutton, David A McAllester, Satinder P Singh, and Joseph Stone. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1050–1056, 1999.