University of Stuttgart

Reinforcement Learning

Policy gradients

Daniel Hennes

26.06.2017

University Stuttgart - IPVS - Machine Learning & Robotics

Policy–based reinforcement learning

- So far we approximated the action–value function and generated a policy from it
- Approximation of the action–value function

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

- Generation of policy by, e.g., ϵ–greedy

$$\hat{q}(s, a, \boldsymbol{w}) \xrightarrow{\epsilon\text{-greedy}} \pi$$

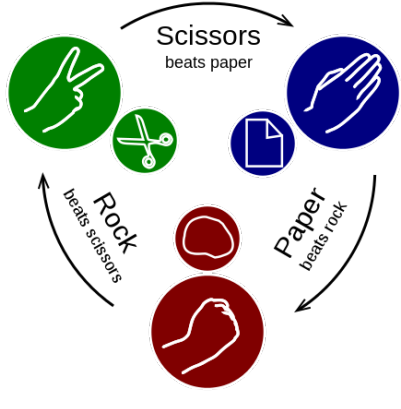
- Now we directly *parameterize* the policy π

Policy–based reinforcement learning

- Advantages:
  - better convergence properties
  - effective in high-dimensional or *continuous* action spaces
  - can learn *stochastic* policies
- Disadvantages:
  - typically converge to a *local* rather than *global* optimum
  - evaluating a policy is typically *inefficient* and *high variance*

Stochastic policies

- Consider the *iterated* version of rock–paper–scissors
- What happens if you play a deterministic policy?
- Which policy is best?



Policy optimization

- Policy optimization:

$$\pi_* = \arg\max_\pi J(\pi)$$

  where J is some *performance measure*

- Discounted return: $G_0 = \sum_{k=0}^{T-1} \gamma^k R_{k+1}$

$$\pi_* = \arg\max_\pi \mathbb{E}_\pi[G_0] = \arg\max_\pi \mathbb{E}_\pi[v_\pi(s_0) \mid S_0 = s_0]$$

- Undiscounted return: $G_0 = \sum_{k=0}^{T-1} R_{k+1}$, i.e., $\gamma = 1$

$$\pi_* = \arg\max_\pi \mathbb{E}_\pi[R_0 + R_1 + \ldots + R_{T-1} \mid S_0 = s_0]$$

- In continuing environments, we could use the average state–value:

$$\sum_s \mu_\pi(s) \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) r(s, a, s')$$

  where μ_π is the steady–state distribution under π

Parameterized policies

- Policies parameterized by parameter $\boldsymbol{\theta} \in \mathbb{R}^d$:
  - deterministic: $a = \pi(s, \boldsymbol{\theta})$
  - stochastic: $a \sim \pi(a \mid s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}\}$
- Objective becomes $\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_0]$
- We can parameterize π in any way, as long as it is *differentiable* wrt to $\boldsymbol{\theta}$
- In general we require $\pi(a \mid s, \boldsymbol{\theta}) \in (0, 1)$ for all s, a
- If the action space is discrete (and not too large): *softmax policy*

$$\pi(a \mid s, \boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

  where $h(\cdot)$ is the *action preference* function
- Can this approach the deterministic policy?
- How does it compare to ϵ-greedy or softmax with action–values?

Policy gradient methods

- Problem:

$$\pi_* = \arg\max_{\pi} \mathbb{E}_{\pi}[R_0 + R_1 + \ldots + R_{T-1} | S_0 = s_0]$$

- Intuition: collect a bunch of trajectories, and ...
  1. Make the good trajectories more probable
  2. Make the good actions more probable
  3. Push the actions towards good actions
- Policy gradient methods:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \widehat{\nabla J(\boldsymbol{\theta})}$$

  - where $\nabla J(\boldsymbol{\theta})$ is the *policy gradient*:

$$\nabla J(\boldsymbol{\theta}) = \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \ldots, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_d} \right)^T$$

  - note that this is approximate gradient *ascent*

Score function gradient estimator

- Consider $\mathbb{E}_{x \sim p(x|\theta)}[f(x)]$ for some score function f
- We want to compute the gradient wrt $\boldsymbol{\theta}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_x[f(x)] &= \nabla_{\boldsymbol{\theta}} \int f(x) p(x \mid \boldsymbol{\theta}) dx \\
&= \int f(x) \nabla_{\boldsymbol{\theta}} p(x \mid \boldsymbol{\theta}) dx \\
&= \int f(x) \frac{\nabla_{\boldsymbol{\theta}} p(x \mid \boldsymbol{\theta})}{p(x \mid \boldsymbol{\theta})} p(x \mid \boldsymbol{\theta}) dx \\
&= \int \left[f(x) \nabla_{\boldsymbol{\theta}} \log p(x \mid \boldsymbol{\theta}) \right] p(x \mid \boldsymbol{\theta}) dx \\
&= \mathbb{E}_x[f(x) \nabla_{\boldsymbol{\theta}} \log p(x \mid \boldsymbol{\theta})]
\end{aligned}
$$

- Note that $\nabla \log x = \frac{\nabla x}{x}$
- Unbiased gradient estimator:
  - sample $x_i \sim p(x \mid \boldsymbol{\theta})$
  - compute sample gradient $f(x_i) \nabla_{\boldsymbol{\theta}} \log p(x_i \mid \boldsymbol{\theta})$

Score function gradient estimator

- Sample gradient: $f(x_i)\nabla_{\boldsymbol{\theta}} logp(x_i \mid \boldsymbol{\theta})$
- $f(x_i)$ is the score of sample x_i
- Moving in the direction of the sample gradient pushes the log–probability of the sample in proportion to its score (*gradient ascent*)
- Valid even if f is:
  - *discontinuous*
  - *unknown*
  - or sample space is a *discrete set*

Score function gradient estimator for policies

- Recall: $\nabla_{\boldsymbol{\theta}} \mathbb{E}_x[f(x)] = \mathbb{E}_x[f(x) \nabla_{\boldsymbol{\theta}} \log p(x \mid \boldsymbol{\theta})]$
- Let us consider a random trajectory τ in place of x:

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$$

and G_0 instead of f, thus we have:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_\tau[G_0] = \mathbb{E}_\tau[G_0 \nabla_{\boldsymbol{\theta}} \log p(\tau \mid \boldsymbol{\theta})$$

$$p(\tau \mid \boldsymbol{\theta}) = \Pr\{S_0\} \prod_{t=0}^{T-1} \pi(A_t \mid S_t, \boldsymbol{\theta}) p(S_{t+1} \mid S_t, A_t)$$

$$\log p(\tau \mid \boldsymbol{\theta}) = \log \Pr\{S_0\} + \sum_{t=0}^{T-1} \log \pi(A_t \mid S_t, \boldsymbol{\theta}) + \log p(S_{t+1} \mid S_t, A_t)$$

$$\nabla_{\boldsymbol{\theta}} \log p(\tau \mid \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{t=0}^{T-1} \log \pi(A_t \mid S_t, \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_\tau[G_0] = \mathbb{E}_\tau \left[G_0 \nabla_{\boldsymbol{\theta}} \sum_{t=0}^{T-1} \log \pi(A_t \mid S_t, \boldsymbol{\theta}) \right]$$

Policy gradient theorem

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[G_0] = \mathbb{E}_{\tau}\left[G_0 \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})\right]$$

$$= \mathbb{E}_{\tau}\left[\left(\sum_{t=0}^{T-1} R_{t+1}\right) \nabla_{\boldsymbol{\theta}} \log \sum_{t=0}^{T-1} \pi(A_t \mid S_t, \boldsymbol{\theta})\right]$$

- For a single reward R_k we have:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[R_k] = \mathbb{E}_{\tau}\left[R_k \sum_{t=0}^{k-1} \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})\right]$$

- Summing over all k, we get:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[G_0] = \mathbb{E}_{\tau}\left[\sum_{k=1}^{T} R_k \sum_{t=0}^{k-1} \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})\right]$$

$$= \mathbb{E}_{\tau}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta}) \sum_{k=t+1}^{T} R_k\right]$$

Policy gradient theorem

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[G_0] &= \mathbb{E}_{\tau}\Big[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta}) \sum_{k=t+1}^{T} R_k\Big] \\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[q_{\pi}(S_t, A_t) \nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta})\right]\end{aligned}$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[q_{\pi}(S_t, A_t) \nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta})\right]$$

Monte–Carlo policy gradient (REINFORCE)

- Update parameters $\boldsymbol{\theta}$ by stochastic gradient ascent
  - using the policy gradient theorem
  - using return G_t as an unbiased sample of $q_\pi(S_t, A_t)$
- Update rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})$$

REINFORCE with baseline

- Monte–Carlo policy gradient still suffers from high variance
- We use a *baseline* to estimate the value function

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$$

- Policy parameters: $\boldsymbol{\theta}$
- Value estimator parameters: $\boldsymbol{w}$
- Update rule:

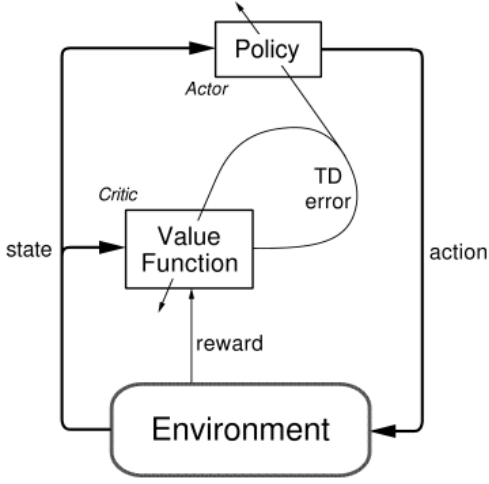
$$\begin{aligned}
\delta &\leftarrow G - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \gamma^t \delta \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})
\end{aligned}$$

- Interpretation:
  - increase log–prob of action A_t proportionally to how much G_t is better than expected
  - baseline accounts for and removes the effect of past actions

Estimating the action–value function

- We are solving the prediction problem: policy evaluation
- How good is policy $\pi_{\boldsymbol{\theta}}$ with current parameters $\boldsymbol{\theta}$?
- Familiar toolset for *fitting* the baseline:
  - Monte–Carlo policy evaluation
  - TD-learning
  - TD(λ)
  - LSPI

Actor-critic concept



Actor-critic vs. baseline

- Actor-critic methods use the value function as a baseline for policy gradients
- Delivers trade off between *variance reduction* of policy gradients with *bias introduction* from value function methods
- **Critic:** updates value–function parameters $\boldsymbol{w}$
- **Actor:** updates policy parameters $\boldsymbol{\theta}$ using critic
- REINFORCE with baseline uses value–function as *baseline* not as a *critic*
  - not used for *bootstrapping*
- One–step actor-critic update:

$$\begin{aligned}
\delta &\leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \gamma^t \delta \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})
\end{aligned}$$

Value functions for the critic

- State-value function: $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$
  - as in one–step Actor–Critic
- Action–value function: $\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$
  - gradient of the policy is the direction that *most improves* q:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}\left[\frac{\partial \hat{q}(s, a, \boldsymbol{w})}{\partial a} \frac{\partial \pi(s, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]$$

- Advantage function: $\hat{a}(s, a) = q_\pi(s, a) - v(s)$
  - e.g., used in Asynchronous Advantage Actor-Critic (A3C)
  - async updates are for speed not for performance

Bias in Actor–Critic algorithms

- Approximating (bootstrapping) the policy gradient introduces bias
- Biased policy gradient might not find the right solution
- But reduces variance and makes learning substantially more efficient
- *Compatible function approximation* avoids this problem:
  - *compatible function approximation theorem*
  - policy gradients are exact

What about continuous action spaces?

- Softmax works for (not too large) discrete action–spaces
- In continuous action spaces, a **Gaussian policy** is a common choice
- Gaussian policy:

$$A_t \sim \mathcal{N}(\mu(S_t, \boldsymbol{\theta}), \sigma^2(S_t, \boldsymbol{\theta}))$$

- Variance may also be constant across the state space
- E.g., neural network outputs the mean of each action dimension as a function of the state

Summary

- *Action–value methods*: learn values, select action accordingly
- *Policy–gradient methods:* learn directly a parameterized Policy
- Advantages:
  - learn specific probabilities for taking the actions
  - learn appropriate levels of exploration, or ...
  - approach deterministic policies
  - can handle continuous action spaces
  - some policies are simpler to represent than value function
  - *policy gradient theorem* (exact expression how performance is affected)
- REINFORCE with baseline reduces variance without adding bias
- Value–functions for bootstrapping (Actor–Critic) introduces bias, ... but substantially reduces variance