Policy Search

Machine Learning (69152)

Rubén Martínez Cantín
Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters θ,

$$V_\theta(x) \approx V^\pi(x)$$
$$Q_\theta(x, a) \approx Q^\pi(x, a)$$

- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrise the policy

$$\pi_\theta(x, a) = p(a|x, \theta)$$

- We will focus again on model-free reinforcement learning

Advantages of Policy-Based RL

Advantages:

- Better convergence properties in complex scenarios
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies (which can be useful in competitive scenarios: self-driving cars, games, etc.)

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Parenthesis: Probabilities and Markov chains

- Remember that for any two random variables[1]:

$$p(a, b) = p(a|b)p(b) \quad , \quad p(a) = \sum_b p(a, b)$$

- And if a and b are independent:

$$p(a, b) = p(a)p(b) \quad , \quad p(a|b) = p(a)$$

- Also, for any function $f(\cdot)$:

$$\mathbb{E}_x[f(x)] = \sum_x f(x)p(x)$$

$$\nabla_\theta \log f(\theta) = \frac{\nabla_\theta f(\theta)}{f(\theta)} \quad \Rightarrow \quad \nabla_\theta f(\theta) = f(\theta)\nabla_\theta \log f(\theta)$$

[1] if b is continuous then, $\sum_b \cdot \Rightarrow \int_b \cdot db$

Parenthesis: Probabilities and Markov chains

- If we have a Markov decision process, with initial probability $p(x_0)$, transition function $p(x_{t+1}|a_t, x_t)$ and policy $\pi(a_t|x_t)$; the probability of an episode $\tau = \{x_0, a_0, x_1, a_1, \ldots, x_n, a_n\}$ is:

$$p(\tau) = p(x_0)\pi(a_0|x_0)p(x_1|x_0, a_0)\ldots\pi(a_{n-1}|x_{n-1})p(x_n|x_{n-1}, a_{n-1})$$

- If we marginalize the actions, we get the probability of the Markov chain $\mathcal{P}$:

$$\begin{aligned}\mathcal{P} = p(x_{0:T}) &= \sum_a p(x_0)\prod_{i=0}^{n}\pi(a_i|x_i)p(x_{i+1}|x_i, a_i)\\ &= \sum_a \pi(a_{0:T}|x_{0:T})p(x_0)\prod_{i=0}^{n}p(x_{i+1}|x_i, a_i)\end{aligned}$$

Objective function

- Goal: given policy $\pi_\theta(x, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- For simplicity, assume an episodic[2] environment with horizon T and $\gamma = 1$:

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_\tau \left[\sum_{t=0}^{T} R(x_t, a_t, x_{t+1}) \right] \\
&= \mathbb{E}_{p(x_0) \prod_{i=0}^{n} \pi(a_i|x_i) p(x_{i+1}|x_i, a_i)} \left[\sum_{t=0}^{T} R(x_t, a_t, x_{t+1}) \right] \\
&= \mathbb{E}_{\mathcal{P}} \left[\sum_{a_{0:T}} \left(\sum_{t=0}^{T} R(x_t, a_t, x_{t+1}) \right) \pi(a_{0:T}|x_{0:T}) \right]
\end{aligned}
$$

[2] It can also work for continuous problems, assuming the MDP is an *aperiodic* and *irreducible* Markov chain, like in MCMC.

Policy Gradient

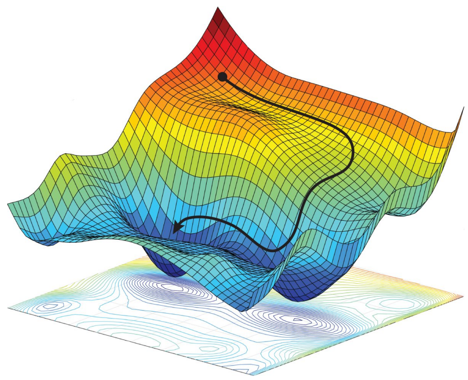
- Policy gradient algorithms search for a *local maximum* in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta)$$

- where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter



Follow the gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\mathcal{P}}\left[\sum_{a_{\mathbf{0}:T}}\left(\sum_{t=0}^{T} R(x_t, a_t, x_{t+1})\right)\nabla_\theta \pi_\theta(a_{0:T}|x_{0:T})\right] \\
&= \mathbb{E}_{\mathcal{P}}\left[\sum_{a_{\mathbf{0}:T}}\left(\sum_{t=0}^{T} R(x_t, a_t, x_{t+1})\right)\pi_\theta(a_{0:T}|x_{0:T})\nabla_\theta \log \pi_\theta(a_{0:T}|x_{0:T})\right] \\
&= \mathbb{E}_{\mathcal{P}}\left[\sum_{a_{\mathbf{0}:T}}\left(\sum_{t=0}^{T} R(x_t, a_t, x_{t+1})\right)\pi_\theta(a_{0:T}|x_{0:T})\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|x_t)\right] \\
&= \mathbb{E}_{\mathcal{P}}\left[\sum_{a_{\mathbf{0}:T}}\pi_\theta(a_{0:T}|x_{0:T})\left(\sum_{t=0}^{T}\left(\sum_{n=0}^{T} R(x_n, a_n, x_{n+1})\right)\nabla_\theta \log \pi_\theta(a_t|x_t)\right)\right] \\
&= \mathbb{E}_{\tau}\left[\sum_{t=0}^{T}\left(\sum_{n=0}^{T} R(x_n, a_n, x_{n+1})\right)\nabla_\theta \log \pi_\theta(a_t|x_t)\right] \\
&= \mathbb{E}_{\tau}\left[\sum_{t=0}^{T}\left(\sum_{n=t}^{T} R(x_n, a_n, x_{n+1})\right)\nabla_\theta \log \pi_\theta(a_t|x_t)\right] \\
&= \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} U_t \nabla_\theta \log \pi_\theta(a_t|x_t)\right]
\end{aligned}
$$

The policy gradient

- We start with the episodic objective:

$$J(\theta) = \mathbb{E}_\tau\left[\sum_{t=0}^{T} R(x_t, a_t, x_{t+1})\right] = \mathbb{E}_\tau\left[U_0\right]$$

- with the following gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[\sum_{t=0}^{T} U_t \nabla_\theta \log \pi_\theta(a_t|x_t)\right]$$

- which can be approximated using Monte Carlo from N episodes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} U_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)}|x_t^{(i)})$$

The policy gradient algorithm

$$U_t = \sum_{n=t}^{T} R(x_n, a_n, x_{n+1})$$

Compute returns

$$\tau = \{x_0, a_0, \ldots, x_n\}$$

Generate data using the policy

Update the policy

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T} U_t \nabla_\theta \log \pi_\theta(a_t|x_t) \right]$$

Softmax policy

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\sum_i \theta_i \phi_i(x,a)$, which using vector notation is $\phi(x,a)^T\theta$.
- Probability of action is proportional to exponentiated weight

$$\boxed{\pi_\theta(a|x) = \frac{e^{\phi(x,a)^T\theta}}{\sum_b e^{\phi(x,b)^T\theta}}}$$

- The score function is

$$\boxed{\nabla_\theta \log \pi_\theta(a|x) = \phi(x,a) - \sum_b \pi_\theta(b|x)\phi(x,b)}$$

Gradient of the softmax

- Probability of action is proportional to exponentiated weight

$$\pi_\theta(a|x) = \frac{e^{\phi(x,a)^T\theta}}{\sum_b e^{\phi(x,b)^T\theta}}$$

- The score function is

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(a|x) &= \nabla_\theta \log e^{\phi(x,a)^T\theta} - \nabla_\theta \log\left(\sum_b e^{\phi(x,b)^T\theta}\right) \\
&= \nabla_\theta \left(\phi(x,a)^T\theta\right) - \frac{\nabla_\theta \sum_b e^{\phi(x,b)^T\theta}}{\sum_b e^{\phi(x,b)^T\theta}} \\
&= \phi(x,a) - \frac{\sum_b \phi(x,b) e^{\phi(x,b)^T\theta}}{\sum_b e^{\phi(x,b)^T\theta}} \\
&= \phi(x,a) - \sum_b \pi_\theta(b|x)\phi(x,b) \\
&= \phi(x,a) - \mathbb{E}_{\pi_\theta}[\phi(x,\cdot)]
\end{aligned}$$

REINFORCE algorithm

- Update parameters by stochastic gradient ascent
- Approximate the episode return $U_t = \sum_{n=t}^{T} R_n$ as an approximation of the expected return.

Algorithm 1 REINFORCE

Initialize θ arbitrarily.
for each episode **do**:
  Generate episode $\tau_R \sim \{x_0, a_0, R_1, x_1, a_1, \ldots x_{n-1}, a_{n-1}, R_n\}$
  for each step t in episode **do**:
    $U_t \leftarrow \sum_{i=t+1}^{T} R_t$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|x_t) U_t$

Stochastic gradient descent

- Remember that the objective is to follow the gradient of $\nabla_\theta J(\theta)$ which requires an expectation over all possible episodes $\mathbb{E}_\tau[F(\theta)]$.
- Instead, we compute a Monte Carlo approximation of the expectation $\widehat{\nabla_\theta J(\theta)}$ based on N episodes $\widehat{\nabla_\theta J(\theta)} \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta J_i(\theta)$.
- But we can update the policy parameters after each episode ($N = 1$)!
- This is a generalization of the *online least squares* method that we saw in the previous lecture, called stochastic gradient descent.

$$\theta \approx \theta - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta J_i(\theta) \approx \theta - \alpha \nabla_\theta J_i(\theta)$$

- where we need to guarantee that we cycle through the values of i. In our case, that we use a different episode each time.

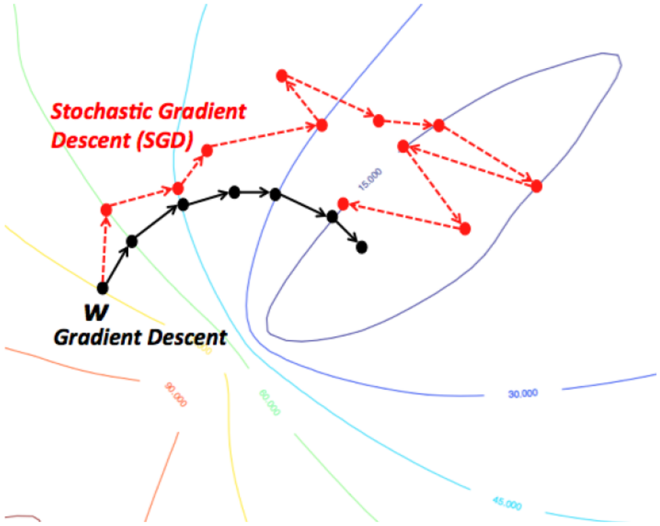
Stochastic gradient descent

Batch gradient descent:

$$\theta = \theta - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta J_i(\theta)$$

Stochastic gradient descent

$$\theta = \theta - \alpha \nabla_\theta J_i(\theta)$$



Stochastic gradient descent has been one of the greatest achievements in machine learning in the past 20 years.

Credit: `https://wikidocs.net/3413`

Alternative Policy Objective Functions

- If we are learning the Q-values, we can also include them in the objective function, using an approximation $\hat{Q}(x, a) \approx Q(x, a)$:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_\tau \left[\sum_{t=0}^{T} Q^{\pi_\theta}(x_t, a_t) \nabla_\theta \log \pi_\theta(a_t | x_t) \right] \\ &\approx \mathbb{E}_\tau \left[\sum_{t=0}^{T} \hat{Q}^{\pi_\theta}(x_t, a_t) \nabla_\theta \log \pi_\theta(a_t | x_t) \right] \end{aligned}$$

- The approximation of $\hat{Q}(x, a)$ can be:
  - Monte Carlo approximation from multiple episodes:
    - Requires perfect resets. Impractical in many cases.
  - Function approximations:
    - For example: with features as we did in *approximate Q-learning* $Q(x, a) = \sum_i w_i \cdot f_i(x, a)$

Actor-Critic algorithm

- Policy $\pi_\theta(a|x) = \phi(x,a)^T\theta$
- Action-value function $Q_w(x,a) = f(x,a)^T w$
- Lower *variance* than Monte-Carlo policy gradient, but might have *bias*.

Algorithm 2 Actor-Critic

Initialize θ and w arbitrarily. Initialize x_0. Sample $a_0 \sim \pi_\theta$.
for each step **do**:
  Sample transition $x_{t+1} \sim p(x_{t+1}|a_t, x_t)$ and collect reward R_{t+1},
  Sample action $a_{t+1} \sim \pi_\theta$
  $\theta = \theta + \alpha_1 \nabla_\theta \log \pi_\theta(a|x) Q_w(x,a)$ ▷ Update policy
  $Qerror = (R_{t+1} + \gamma \max_{a'} Q(x_{t+1}, a')) - Q(x,a)$
  $w = w + \alpha_2 Qerror \cdot f(x,a)$ ▷ Update Q-function

Policy Optimization

- Policy search is an optimization problem
  - Find θ that maximizes $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing / Local search
  - Evolutionary algorithms (e.g.: genetic)
  - Bayesian optimization
- Evolutionary and Bayesian approaches find global optimum.
- Gradient methods are more efficient for high dimensional problems.
  - Deep neural networks allow high dimensional parametric policies and value functions.
  - Open research to get the best of both worlds.

Global vs local optimization

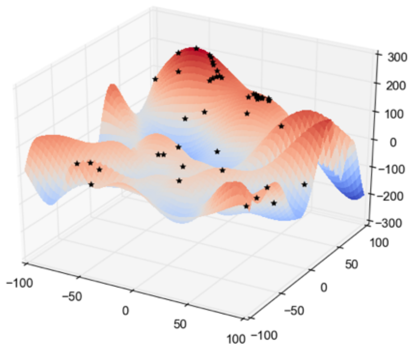


Figure: Bayesian optimization

Example (Video):
Resilient robot

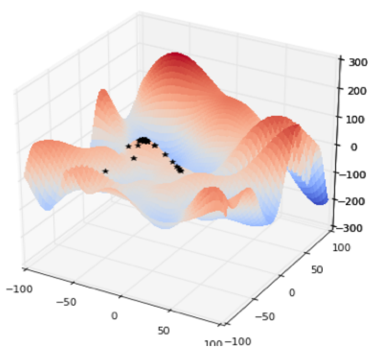


Figure: Gradient method/local search

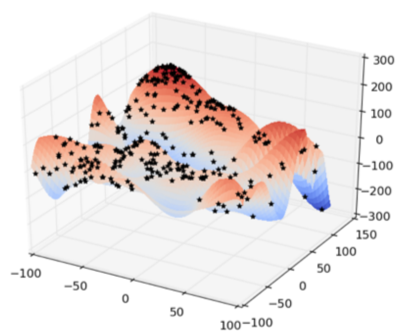


Figure: Evolutionary algorithm