Direct Learning of the Policy Function

Joseph Le Roux

10/07/26

Outline

- Policy Gradient: Motivation
- REINFORCE aka Policy Gradient Algorithm
- Improvements of Policy Gradient
- Recap

1

Policy Gradient: Motivation

1 Summary/Motivation

Recall we want to learn the optimal policy.

What we do

- we learn state-action values $q(s, a)$ via DP, MC, TD...
- and then we can deduce the best policy
- 🤯 quite a convoluted road!

What we will do

- try to learn the policy directly!
- learn a stochastic policy (while we explore/exploit + differentiable)
- and set the final policy to the greedy approximation (or not, we'll experiment in lab)

1 Motivation (2)

QLearning/SARSA

Too complex: we do not always need to compute the exact values of $q(s, a)$ to decide what the best action is.

Other bottlenecks

Continuous actions Q-learning requires $\max_a Q(s, a)$, which is intractable for continuous action spaces (cf. mountain car)

Stochastic policies some environments require stochastic optimal policies (when some info is missing, cf. INFO3)

Direct parametrization policy class can encode knowledge (e.g. Gaussian Distribution if this is the underlying physical model)

Policy (approximation) function

- Same idea here that with the function approximation for q function (functions are better than tables)
- The policy function can be richly parametrised $\longrightarrow$ better prediction of actions

2

REINFORCE aka Policy Gradient Algorithm

A general policy function for finite action sets

Gibbs distribution (exponential family, log-linear model...)

Given a *score function* $\sigma : A \times S \longrightarrow \mathbb{R}$:

$$\pi(a|s) = \frac{\exp(\sigma(a,s))}{\sum_{a'} \exp(\sigma(a',s))}$$

In other words: we compute each action's score, store them in a vector, and perform softmax. (think $\sigma = q$)

Definition of σ

As for $\hat{q}$, a parameterized function which can be linear, convex, polynomial, implementable by a neural network...
$\Longrightarrow$ we will use a linear function parametrised by $\mathbf{w}$

$$\sigma(a,s;\mathbf{w}) = \sum_{i=1}^{d} \mathbf{w}_i^a \times x(s)_i = {}^a \cdot x(s)$$

2 Characterise policy function (1)

Learning Objective

We still want a policy that earns high expected gains, *i.e.* that generates good trajectories from initial state s. We want to find parameters $\mathbf{w}$ which maximize:

$$J(\mathbf{w}) = \mathbb{E}_{\tau \sim \pi}[G_0 | S_0 = s]$$

i.e. we want to maximise the expected gain of the initial state.

- $\tau \sim \pi$ means that τ is generated from π and s (do not forget p_T)
- We will learn the optimal parameters by gradient ascent (maximisation!)

2 Characterise policy function (2.1)

Learning Objective

we want to maximise the expected gain of the initial state.

$$J() = \mathbb{E}_{\tau\sim\pi}[G|S_0 = s] = \sum_{\tau} p(\tau|S_0 = s)G_0(\tau) \quad \text{(def. expectation)}$$

where $G_i(\tau)$ is the expected gain of state s_i in trajectory τ

What is $p(\tau|s_0)$?

Assume $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n$. Use chain rule:

$$\begin{aligned} p(\tau|s_0) &= p(s_0, a_0, r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0) \\ &= p(s_0|s_0) \times p(a_0, r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0, s_0) \\ p(\tau|s_0) &= p(a_0, r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0) \end{aligned}$$

2 Characterise policy function (2.2)

Learning Objective

we want to maximise the expected gain of the initial state.

$$J() = \mathbb{E}_{\tau\sim\pi}[G|s_0] = \sum_{\tau} p(\tau|s_0)G_0(\tau) \quad \text{(def. expectation)}$$

where $G_i(\tau)$ is the gain of state s_i in trajectory τ

What is $p(\tau|s_0)$?

$$\begin{aligned}
p(\tau|s_0) &= p(a_0, r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0) \\
&= p(a_0|s_0) \times p(r_1, s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0, a_0) \\
&= \underbrace{p(a_0|s_0)}_{\text{this is } \pi} \times \underbrace{p(r_1|s_0, a_0)}_{=1 \text{ for our MDPs}} \times p(s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0, a_0, r_1) \\
&= \pi(a_0|s_0) \times p(s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0, a_0, \underbrace{r_1}_{\text{not in } p_{\tilde{\tau}}, \pi}) \\
&= \pi(a_0|s_0) \times p(s_1, a_1, r_2, s_2 \dots a_{n-1}, r_n, s_n|s_0, a_0) \\
&= \pi(a_0|s_0) \times p(s_1, a_1, s_2 \dots a_{n-1}, s_n|s_0, a_0) \quad \text{(can remove all rewards)}
\end{aligned}$$

2 Characterise policy function (2.3)

Learning Objective

we want to maximise the expected gain of the initial state.

$$J() = \mathbb{E}_{\tau\sim\pi}[G|s_0] = \sum_{\tau} p(\tau|s_0)G_0(\tau) \quad \text{(def. expectation)}$$

where $G_i(\tau)$ is the gain of state s_i in trajectory τ

What is $p(\tau|s_0)$?

$$\begin{aligned}
p(\tau|s_0) &= \pi(a_0|s_0) \times p(s_1, a_1, s_2 \dots a_{n-1}, s_n|s_0, a_0) \\
&= \pi(a_0|s_0) \times p(s_1|s_0, a_0) \times p(a_1, s_2 \dots a_{n-1}, s_n|s_0, a_0, s_1) \\
&= \pi(a_0|s_0) \times p_T(s_1|s_0, a_0) \times p(a_1, s_2 \dots a_{n-1}, s_n|s_0, a_0, s_1) \\
&= \pi(a_0|s_0) \times p_T(s_1|s_0, a_0) \times \underbrace{p(a_1|s_0, a_0, s_1)}_{\text{use Markov hyp.}} \times p(s_2 \dots a_{n-1}, s_n|s_0, a_0, s_1, a_1) \\
&= \pi(a_0|s_0) \times p_T(s_1|s_0, a_0) \times \pi(a_1|s_1) \times p(s_2 \dots a_{n-1}, s_n|s_0, a_0, s_1, a_1) \\
&= \Big(\prod_{i=0}^{1} \pi(a_i|s_i)\Big) \times p_T(s_1|s_0, a_0) \times p(s_2 \dots a_{n-1}, r_n, s_n|s_0, a_0, s_1, a_1)
\end{aligned}$$

2 Characterise policy function (2.4)

Learning Objective

we want to maximise the expected gain of the initial state.

$$J() = \mathbb{E}_{\tau\sim\pi}[G|s_0] = \sum_{\tau} p(\tau|s_0)G_0(\tau) \quad \text{(def. expectation)}$$

where $G_i(\tau)$ is the gain of state s_i in trajectory τ

What is $p(\tau|s_0)$?

$$\begin{aligned}
p(\tau|s_0) &= \Big(\prod_{i=0}^{1}\pi(a_i|s_i)\Big) \times p_T(s_1|s_0,a_0) \times p(s_2 \dots a_{n-1}, r_n, s_n|s_0,a_0,s_1,a_1) \\
&= \Big(\prod_{i=0}^{1}\pi(a_i|s_i)\Big) \times p_T(s_1|s_0,a_0) \times \underbrace{p(s_2|s_0,a_0,s_1,a_1)}_{\text{Markov hyp.}} \times p(\dots|s_0,a_0,s_1,a_1,s_2) \\
&= \Big(\prod_{i=0}^{1}\pi(a_i|s_i)\Big) \times p_T(s_1|s_0,a_0) \times p_T(s_2|s_1,a_1) \times p(\dots|s_0,a_0,s_1,a_1,s_2) \\
&= \Big(\prod_{i=0}^{1}\pi(a_i|s_i)\Big) \times \Big(\prod_{i=0}^{1}p_T(s_{i+1}|s_i,a_i)\Big) \times p(\dots|s_0,a_0,s_1,a_1,s_2)
\end{aligned}$$

2 Characterise policy function (2.5)

Learning Objective

we want to maximise the expected gain of the initial state.

$$J() = \mathbb{E}_{\tau\sim\pi}[G|s_0] = \sum_{\tau} p(\tau|s_0)G_0(\tau) \quad \text{(def. expectation)}$$

where $G_i(\tau)$ is the gain of state s_i in trajectory τ

What is $p(\tau|s_0)$?

$$\begin{aligned} p(\tau|s_0) &= \Big(\prod_{i=0}^{1} \pi(a_i|s_i)\Big) \times \Big(\prod_{i=0}^{1} p_T(s_{i+1}|s_i, a_i)\Big) \times p(\ldots|s_0, a_0, s_1, a_1, s_2) \\ &= \ldots \text{ same tricks for all positions } 3, 4, \ldots, n \\ &= \Big(\prod_{i=0}^{n-1} \pi(a_i|s_i)\Big) \times \Big(\prod_{i=0}^{n-1} p_T(s_{i+1}|s_i, a_i)\Big) \\ &= \prod_{i=0}^{n-1} \big(\pi(a_i|s_i) p_T(s_{i+1}|s_i, a_i)\big) \end{aligned}$$

2 Characterise policy function (3)

We want to learn via gradient ascent: what is the gradient of J?

Gradient of the Objective $J()$

$$
\begin{aligned}
\nabla J() &= \nabla \mathbb{E}_{\tau\sim\pi}[G_0|s] \\
&= \nabla \sum_{\tau} p(\tau|s) G_0(\tau) \quad \text{(def. expectation)} \\
&= \sum_{\tau} \nabla p(\tau|s) G_0(\tau) \quad \text{(gradient} \leftrightarrow \text{sum)} \\
&= \sum_{\tau} G_0(\tau) \nabla p(\tau|s) \quad \text{(gain is constant)} \\
&= \sum_{\tau} \frac{p(\tau|s_0)}{p(\tau|s)} G_0(\tau) \nabla p(\tau|s) \quad \text{(multiply by one)} \\
&= \sum_{\tau} p(\tau|s) G_0(\tau) \frac{\nabla p(\tau|s)}{p(\tau|s)} \quad \text{(rearrange)} \\
&= \sum_{\tau} p(\tau|s) G_0(\tau) \nabla \log p(\tau|s) \quad \text{(log trick): } \nabla \log f = \nabla f / f \\
&= \mathbb{E}_{\tau\sim\pi}[G_0(\tau) \nabla \log p(\tau|s)] \quad \text{(def. expectation)}
\end{aligned}
$$

2 Characterise policy function (4)

Gradient of the Objective $J()$

$$\begin{aligned}
\nabla J() &= \mathbb{E}_{\tau\sim\pi}\Big[G_0(\tau)\nabla\log\big(p(\tau|s_0)\big)\Big] \\
&= \mathbb{E}_{\tau\sim\pi}\Big[G_0(\tau)\nabla\log\big(\prod_i p_T(s_{i+1}|s_i,a_i)\pi(a_i|s_i)\big)\Big] \\
&= \mathbb{E}_{\tau\sim\pi}[G_0(\tau)\Big(\sum_i\big(\nabla\log(p_T(s_{i+1}|s_i,a_i))\big)+\big(\nabla\log(\pi(a_i|s_i))\big)\Big)] \qquad p_T(s_{i+1}|s_i,a_i) \text{ does not depend on } \sigma \\
&= \mathbb{E}_{\tau\sim\pi}[G_0(\tau)\Big(\sum_i\nabla\log\big(\pi(a_i|s_i)\big)\Big)]
\end{aligned}$$

Conclusion: $J()$ has the same gradient (including same zeros) than function:

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{\tau\sim\pi}[G_0(\tau)\sum_i\log\big((\pi(a_i|s_i)\big)]$$

so the (local) maximums are the same, we can optimise it instead! (or equivalent, use $\nabla\hat{J}$ as an estimate of ∇J)

2 Characterise policy function (5)

Our new objective

We want to maximize:

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{\tau\sim\pi}[G_0(\tau)(\sum_i \log(\pi(a_i|s_i)))]$$

Intuition

- maximise the (log-)likelihood of the state-actions in trajectories but weight them with state gains earned by the trajectory.
  - $\Longrightarrow$ arrow actions appearing in high-gain trajectories will be weighted more $\longrightarrow$ will become more probable to maximise objective.

Intuition 2: Conection with supervised learning and classification

Same objective as classification (cross-entropy, log-likelihood), with the addition of weights

1. High-gain means *positive examples*
2. Low-gain means *negative examples*

2 Characterise policy function (6)

But... how can we compute the expectation?

Monte-Carlo again: approximate expectation with sampling

- We will use m different trajectories $\tau^{(1)} ... \tau^{(m)}$ obtained by running the current policy in the same environment:

$$\hat{J}(\mathbf{w}) \approx \frac{1}{m} \sum_{k=1}^{m} G_0(\tau^{(k)})(\sum_i \log(\pi(a_i^{(k)}|s_i^{(k)})))$$

- Approximation is exact when $m \longrightarrow \infty$

Approximate with one sample ($m = 1$)

Coarse approximation but it works$^{\text{TM}}$!

$$\hat{J}(\mathbf{w}) \approx G_0(\tau)(\sum_i \log(\pi(a_i|s_i)))$$

where τ is a trajectory returned by the agent with its current policy

2 Policy Gradient Algorithm (REINFORCE)

Mix Monte-Carlo, policy gradient, and stochastic gradient descent/ascent:

Reinforce

- Input:
  - initial parameters
  - stepsizes α_k
- Loop for $k = 1 \longrightarrow K$
  - from initial state s_0 generate trajectory $\tau = s_0, a_0, r_1 \ldots, a_{n-1}, r_n, s_n$
  - compute gain G_0
  - $\longleftarrow + \alpha_k G_0 (\sum_i \nabla \log(\pi(a_i|s_i)))$

One missing ingredient: what is $\nabla \log(\pi$ (a|s))??

2 Compute Gradient in Policy Gradient (1)

Compute $\nabla \log(\pi(a_i|s_i))$

- Remember

$$\pi(a|s) = \frac{\exp(\sigma(a,s))}{\sum_{a'} \exp(\sigma(a',s))}$$

- and

$$\sigma(a,s;\mathbf{w}) = \sum_{i=1}^{d} \mathbf{w}_i^a \times s_i = {}^a \cdot s$$

2 Compute Gradient in Policy Gradient (2)

(we use s instead of $x(s)$ in the proof)

$$
\begin{aligned}
\nabla \log \frac{\exp(a_i \cdot s_i)}{\sum_a \exp(a \cdot s_i)} &= \nabla\Big(\log\big(\exp(a_i \cdot s_i)\big) - \log\big(\sum_a \exp(a \cdot s_i)\big)\Big) \\
&= \nabla\Big(\log\big(\exp(a_i \cdot s_i)\big)\Big) - \nabla\Big(\log\big(\sum_a \exp(a \cdot s_i)\big)\Big) \\
&= \nabla(a_i \cdot s_i) - \nabla\Big(\log\big(\sum_a \exp(a \cdot s_i)\big)\Big) \\
&= \nabla(a_i \cdot s_i) - \frac{\sum_a \nabla \exp(a \cdot s_i)}{\sum_{a'} \exp(a' \cdot s_i)} \\
&= \nabla(a_i \cdot s_i) - \frac{\sum_a \exp(a \cdot s_i) \nabla(a \cdot s_i)}{\sum_{a'} \exp(a' \cdot s_i)} \\
&= \nabla(a_i \cdot s_i) - \sum_a \nabla(a \cdot s_i) \frac{\exp(a \cdot s_i)}{\sum_{a'} \exp(a' \cdot s_i)} \\
&= \nabla(a_i \cdot s_i) - \sum_a \nabla(a \cdot s_i) \pi(a|s_i) \\
&= \nabla(a_i \cdot s_i) - \mathbb{E}_{a \sim \pi}[\nabla(a \cdot s_i)]
\end{aligned}
$$

2 Compute Gradient in Policy Gradient (3)

Summing up previous slide

$$\begin{aligned}\nabla \log \pi(a_i|s_i) &= \nabla(_{a_i} \cdot s_i) - \mathbb{E}_{a\sim\pi}[\nabla(_a \cdot s_i)] \\ &= \nabla(_{a_i} \cdot s_i) - \sum_j \pi(a_j|s_i)\nabla(_{a_j} \cdot s_i)\end{aligned}$$

Practical computation

This gives for each parameter k of $_a, \forall a$:

$$\frac{\partial \log \pi(a_i|s_i)}{\partial_a} = \begin{cases} s_i - \pi(a_i|s_i) \times s_i & \text{if } a = a_i, \\ -\pi(a|s_i) \times s_i & \text{otherwise.} \end{cases}$$

That we can rewrite as:

$$\frac{\partial \log \pi(a_i|s_i)}{\partial_a} = s_i\big(\mathbf{1}[a = a_i] - \pi(a|s_i)\big)$$

where $\mathbf{1}[expr]$ is 1 if *expr* is `True` and 0 otherwise

2 Back to the algorithm

Input:

- initial parameters (matrix $|A| \times d$, where d is the number of parameters for each state)
- stepsizes α
- matrix `grad`: same dimensions as

Loop for $k = 1 \longrightarrow K$

- `grad` $\longleftarrow 0$
- from initial state s_0 generate trajectory $\tau = s_0, a_0, r_1 \ldots, a_{n-1}, r_n, s_n$
- compute gain G
- for each position i:
  - `grad`$[a_i]$ += s_i
  - for each action a:
    - `grad[a]` -= $\pi(a_i|s_i) \times s_i$
- $\longleftarrow + \alpha \times G \times$ `grad`

3 Improvements of Policy Gradient

3 Problems with Naive Policy gradient

Variance

G can (and will) vary a lot

- No Bellman $\longrightarrow$ use gains of 1 sampled trajectory
- big variance due to sampling and stochasticity in π and T

Coarse weighting

- weight G_0 for all actions in a trajectory $\longrightarrow$ slow training
- maybe some actions were good, some actions were bad
- how can we individualize weights of each action?

Gain at each step

Intuition: action at step i should not be penalised by decisions that were made before i

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{\tau\sim\pi}[(\sum_i \gamma^i G_i \log(\pi(a_i|s_i)))]$$
$$= \mathbb{E}_{\tau\sim\pi}[(\sum_i \gamma^i (\sum_{j=i} \gamma^{j-i} r_j) \log(\pi(a_i|s_i)))]$$

Proof

a bit technical, see OpenAI site

3 PG with individual gains

Input:

- initial parameters (matrix $|A| \times d$, where d is the number of parameters for each state)
- stepsizes α
- matrix `grad`: same dimensions as

Loop for $k = 1 \longrightarrow K$

- `grad` $\longleftarrow 0$
- from initial state s_0 generate trajectory $\tau = s_0, a_0, r_1 \ldots, a_{n-1}, r_n, s_n$
- compute gains $G_0 \ldots G_{n-1}$
- for each position i
  - `grad`$[a_i]$ += $\gamma^i G_i \times s_i$
  - for each action a:
    - `grad[a]` -= $\gamma^i G_i \times \pi(a_i|s_i) \times s_i$
- $\longleftarrow + \alpha_k \times$ `grad`

3 Baseline

we can *rescale* gains (towards zero) and still get a policy gradient (proof. Pieter Abeel, blackboard)

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{\tau\sim\pi}[(\sum_i \gamma^i (G_i - b_i) \log(\pi(a_i|s_i)))]$$

- b_j is called a baseline and can depend on the state (or not) but not on action
- a simple but good baseline is $b = (G_0 + \cdots + G_{n-1})/n$ the average gain
  - intuition: we want to push up the best half of actions and push down the other half.
- Can we have better baseline ?

3 PG with baseline

Input:

- initial parameters (matrix $|A| \times d$, where d is the number of parameters for each state)
- stepsizes α_k
- matrix `grad`: same dimensions as

Loop for $k = 1 \longrightarrow K$

- `grad` $\longleftarrow 0$
- from initial state s_0 generate trajectory $\tau = s_0, a_0, r_1 \ldots, a_{n-1}, r_n, s_n$
- compute gains $G_0 \ldots G_{n-1}$
- compute $\widehat{G} = \frac{1}{n} \sum_i G_i$
- for each action index i in τ:
  - `grad`$[a_i]$ += $\gamma^i (G_i - \widehat{G}) \times s_i$
  - for each action a
    - `grad[a]` -= $\gamma^i (G_i - \widehat{G}) \times \pi(a_i|s_i) \times s_i$
- $\longleftarrow$ $\theta + \alpha \times$ `grad`

3 Baseline (2)

A better baseline is

$b_j = \mathbb{E}[G_j|s_j]$ the expected (average) gain for state s_j (state values $v(s_j)$!). Intuition:

- $G_j = b_j$: the actual gain is as we expected in this state, we do not need to update
- $G_j > b_j$: the actual gain is better than we expected in this state, we need to increase the probability of the action
- $G_j < b_j$: the actual gain is worse than we expected in this state, we need to decrease the probability of the action

But... how do we know/compute the expected gain?

- we estimate it! Like we did for SARSA/Q-Learning: we minimise MSE

3 PG with Bellman baseline

Input:

- initial parameters for σ, initial parameters ω for $\hat{v}$, step size α
- matrix pgrad: same dimensions as , vgrad same dimensions as ω

Loop for $k = 1 \longrightarrow K$

- pgrad $\leftarrow$ 0, vgrad $\leftarrow$ 0
- from initial state s_0 generate trajectory $\tau = s_0, a_0, r_1 \ldots, a_{n-1}, r_n, s_n$
- compute estimated expectated gains $\hat{v}(s_0, \omega) \ldots \hat{v}(s_{n-1}, \omega)$
- for each index i in τ:
  - $\tilde{G}_i = r_{i+1} + \gamma \hat{v}(s_{i+1}, \omega) - v(s_i, \omega)$
  - for each parameter index p in $_{a_i}$
    - if $a = a_i$, grad[a][p] += $\gamma^i \tilde{G}_i \times (s_i[p] - \pi(a_i|s_i) \times s_i[p])$
    - else grad[a][p] -= $\gamma^i \tilde{G}_i \times \pi(a_i|s_i) \times s_i[p]$
  - vgrad += $(r_{i+1} + \gamma \hat{v}(s_{i+1}, \omega) - \hat{v}(s_i, \omega)) \times s_i$
- $\leftarrow + \alpha \times$ pgrad ; $\omega \leftarrow \omega + \alpha \times$ vgrad

What is this baseline ?

$$\tilde{G}_i = r_{i+1} + \gamma \hat{v}(s_{i+1}, \omega) - \hat{v}(s_i, \omega)$$

is the TD error!

- Similar to the *advantage* (see MDP slides)

Actor-Critic

This algorithm has 2 predictions:

1. π, the actor (performing actions)
2. $\hat{v}$, the critic (judgeing gains received)

4
Recap

4 To sum up

Model policy function

using linear function and softmax

Learn policy function parameters

- by gradient descent
- maximise gains for initial state as objective
- compute/simplify the gradient for this loss
- a weighted version of likelihood as loss to:
  - increase probabilities for action followed by high gains
  - decrease probabilities followed by low gains

Reduce variance due to sampling by using baselines

- simple baselines (average gains)
- Bellman baselines $\longrightarrow$ Actor-Critic algorithm
  - mix TD and policy gradient!
  - the base of many state-of-the-art RL algorithms