Reinforcement Learning

Aldo Faisal with contributions
by Ed Johns and Paul Bilokon

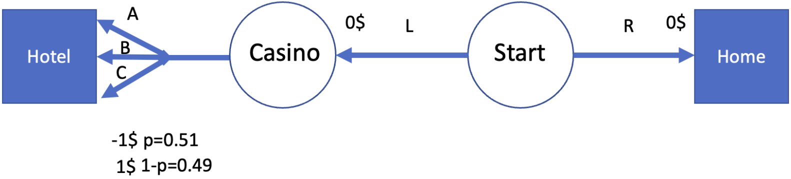
Imperial College London

October 2022 – Version 8.3

Section overview

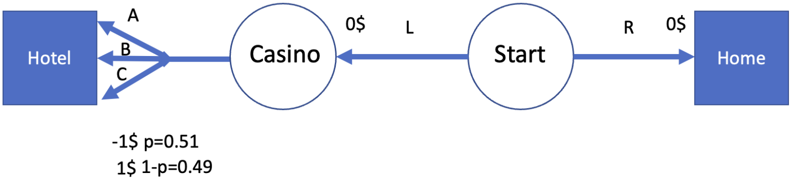
Deep Q Learning

Maximisation Bias I



Consider the following MDP
Going right towards the terminal Home state gives a reward of 0. Going left gives via the Casino state a reward of 0 and a variable reward irrespective of actions A,B,C of $+1$ or -1 with a biased coin-flip. The average reward from the Casino is thus $+1 \times 0.49 + (-1) \times 0.51 = -0.02$ no matter what action we chooose and we invariable end up in the hotel.

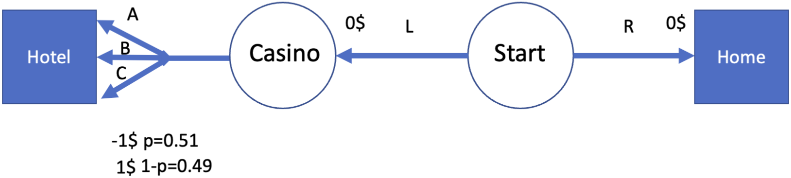
Maximisation Bias I



Thus, in the Start state the expectation is (assuming no discounting) that choosing R over L is marginally beneficial ($0 > -0.02$). However, the large variance in the reward following action L confuses the Q learner. For example, we may perfectly well see rewards episodes in Casino for choosing action$r(Casino, A, Hotel) = -1, r(Casino, B, Hotel) = +1, r(Casino, A, Hotel) = -1$, while reward episodes for Home always look like $r(Start, Right, Home) = 0$.

Maximisation Bias 2 I

Consider the following MDP



How does a Q Learning agent learn?

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a') - Q(s_t, a)\right] \tag{61}$$

We update with a max over actions of the successor state. So when update the value for Q(Start,Left) we perform a max over actions in successor state Casino and immediately note down the

Maximisation Bias 2 II

$+1$ reward from choosing B. In the limit of large number of trials all actions will yield an average reward of -0.02, but that is only asymptotically reached. This illustrates a more general relationship (that we do not prove), namely that when we take a max over all actions we will always somewhat overestimate values.

Double Q Learning (van Hasselt et al, 2017, AAAI) I

Situation: In regular Deep Q-learning we use the target network for two tasks:

1. Identify action with highest Q value,
2. Determine Q value of this action

Complication: The maximum Q value may be overestimated (variance-bias problem), because an unusually high value from the main network Q does not mean that there is an unusually high value from the target network Q'.
The problem with (Deep) Q-Learning is that the same samples (i.e. the Q network) are being used to decide which action is the best (highest expected reward), and the same samples are also being used to estimate that specific action-value.
Solution: In Double Q-Learning (van Hasselt et al, 2017, AAAI) we separate the two estimates: We use a target network Q' for 1. but

Double Q Learning (van Hasselt et al, 2017, AAAI) II

use the regular Q for 2. (or the converse). Thus, we make the selection of the action with highest Q value, and selection of the Q value used in Bellman updates become independent from each other (different networks) so we become less susceptible to variance in each estimate. This reduces the frequency by which the maximum Q value may be overestimated, as it is less likely that both the networks are overestimating the same action.

Actors & Critics

- Value Based methods
  - Find/approximate optimal value function (mapping action to value).
  - These are considered more sample efficient and steady.
  - Examples: Q Learning, Deep Q Networks, Double Dueling Q Networks, etc
- Policy-Based methods
  - Find the optimal policy directly without the Q/V-function.
  - Policy-based are considered to have a faster convergence and are better for continuous and stochastic environments.
  - Examples: Policy-Based algorithms like Policy Gradients and REINFORCE

Section overview

Policy Gradients

Policy-Based methods I

- Learn directly the policy without using an intermediate V/Q function learning approach
- We look directly at a parametrised policy π that depends on parameters θ
- Probability to observe a trace depends on the parameters of the policy

$$p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{62}$$

Policy-Based methods II

- Optimal policy parameters θ^* are defined via maximum average return

$$\theta^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]_{\tau \propto p_\theta(\tau)} \tag{63}$$

Policy gradients I

- Before: Learn the values of actions and then select actions based on their estimated action-values. The policy was generated directly from the value function
- We want to learn a parameterised policy that can select actions without consulting a value function. The parameters of the policy are called policy weights
- A value function may be used to learn the policy weights but this is not required for action selection
- Policy gradient methods are methods for learning the policy weights using the gradient of some performance measure with respect to the policy weights
- Search directly for the optimal policy π^* by optimising policy parameters θ[5]

Policy gradients II

- Policy gradient methods seek to maximise performance and so the policy weights are updated using gradient ascent
- Recall that the optimal policy is the policy that achieves maximum expected (future) return
- Find $\theta^* = \arg\max_\theta \mathbb{E}\left[\sum_t r(s_t, a_t)\right]_{\tau \sim p_\theta(\tau)}$ (infinite horizon case) or $\theta^* = \arg\max_\theta \sum_{t=0}^{T} \mathbb{E}\left[r(s_t, a_t)\right]_{(s_t, a_t) \sim p_\theta(\tau)}$ (finite horizon case)
- We can use any parametric supervised machine learning model to learn policies $\pi(a|a;\theta)$ where θ represents the learned parameters

[5] Note: θ is a parameter of the policy, before we had $\mathbf{w}$ as a parameter of the value function

Finite Difference Policy Gradient I

How to compute the gradient of the policy?
Setting up notation first:

- Policy weight vector θ
- The policy is $\pi(a|s, \theta)$, which represents the probability that action a is taken in state s with policy weight vector θ
- Performance measure $J(\theta)$ in the episodic case: $J(\theta) = V^{\pi}(s_0)$
- Performance is defined as the value of the start state under the parameterised policy (a cost-to-go).

Finite Difference Policy Gradient I

Finite differentiation of the policy by performing gradient ascent using the partial derivative:

- Compute the partial derivative of the performance measure $J(\theta)$ with respect to θ using finite difference gradient approximation
- Compute $J(\theta)$
- Perturb parameters θ by small positive amount (ϵ) in kth dimension and compute $J(\theta + uk\epsilon)$ where u_k is unit vector that is 1 in kth component and 0 elsewhere
- The partial derivative is approximated by finite differences

$$\frac{(J(\theta + uk\epsilon) - J(\theta)}{\epsilon}$$

Finite Difference Policy Gradient II

- Update role $\theta_k = \theta_k + \alpha(\frac{(J(\theta+uk\epsilon)-J(\theta)}{\epsilon}))$

Simple, noisy and inefficient procedure that is sometimes effective, but This procedure works for arbitrary policies (can be non-differentiable).

Direct Policy gradients I

Let us try a direct derivation of the policy gradients (Levine's derivation)

$$\theta^* = \arg\max_\theta \mathbb{E}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]_{\tau \propto p_\theta(\tau)} \tag{64}$$

Average return is approximated by empirical mean over N traces

$$J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]_{\tau \propto p_\theta(\tau)} \approx \frac{1}{N}\sum_{i=i}^{N}\sum_{t} r(s_{i,t}, a_{i,t}) \tag{65}$$

Direct Policy Gradients: Log-Grad trick

We will first need to setup the so called Log-Grad trick:
Note that an expression ${}_\theta \log \pi_\theta(\tau)$ has the structure $\frac{\partial}{\partial x} f(x)$

$$\frac{partial}{\frac{}{\partial x} f(x) = \frac{f(x)}{f(x)} \frac{partial}{\frac{}{partialx} f(x) = f(x) \frac{\frac{partial}{partialx} f(x)}{f(x)}}} \tag{66}$$

$$\pi_\theta(\tau)_\theta log \pi_\theta(\tau) \tag{67}$$

Direct Policy Gradients: Derivation 1

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity

$$\underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)} = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \underline{\nabla_\theta \pi_\theta(\tau)}$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)}] = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)} r(\tau) d\tau = \int \underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)} r(\tau) d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

Direct Policy gradients: Derivation 2

[Space for writing]

Direct Policy Gradients: Derivation 3

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \underline{\log \pi_\theta(\tau)} r(\tau)]$$

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

$$\log \pi_\theta(\tau) = \underline{\log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}$$

$$\nabla_\theta \overbrace{\left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}\right]}$$

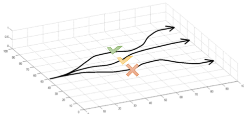
$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

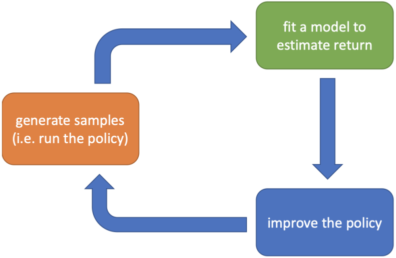
Direct Policy Gradients: Derivation 4

[Space for writing]

Direct Policy Gradients: Derivation 5

- A direct implementation of this result is the classic Policy Gradient algorithms, REINFORCE





REINFORCE algorithm I

- Before we had to learn a value function through function approximation and then derive a corresponding policy
- Often learning a value function can be intractable (more unstable / was, at the time, intractable for large state spaces).
- REINFORCE provided a way to directly optimize policies to get around this problem.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

- The method suffer from high variance in the sampled trajectories, thus stabilising model parameters is difficult.

REINFORCE algorithm II

- Any erratic trajectory can cause a sub-optimal shift in the policy distribution.
- Solutions involve the introduction of a baseline or actor-critics.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

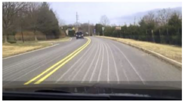
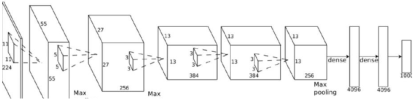
Policy gradients vs Maximum Liklelihood

$$\text{recall: } J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

What does this policy term really mean?
Lets decompose

- Predicting the action given the state can be seen as a supervised learning problem $a = f(s)$.

Gaussian Distribution I

The Gaussian distribution is the most important probability distribution for continuous-valued random variables. We will be adopting a common, but mathematically slightly sloppy, language and call the "probability density function" a "distribution".

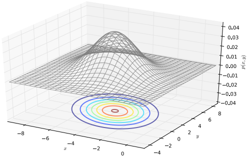


Figure: Multivariate Gaussian distribution of two random variables x, y.

Gaussian Distribution II

The **multivariate Gaussian distribution** is fully characterized by a **mean vector** $\boldsymbol{\mu}$ and a **covariance matrix** $\boldsymbol{\Sigma}$ and defined as

$$p(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}\exp\left(-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{68}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is a random variable. We write $\boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{x}\,|\,\boldsymbol{\mu},\,\boldsymbol{\Sigma}\right)$ or $\boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{\mu},\,\boldsymbol{\Sigma}\right)$. Figure 1 shows a bi-variate or 2-D Gaussian (mesh), with the corresponding contour plot.

Lets spell out the elements of the 2-D Gaussian distribution of two random variables $\boldsymbol{x}, \boldsymbol{y}$, where $\boldsymbol{\Sigma}_{xx} = \mathrm{Cov}[\boldsymbol{x},\boldsymbol{x}]$ and $\boldsymbol{\Sigma}_{yy} = \mathrm{Cov}[\boldsymbol{y},\boldsymbol{y}]$ are the marginal covariance matrices of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, and $\boldsymbol{\Sigma}_{xy} = \mathrm{Cov}[\boldsymbol{x},\boldsymbol{y}] = \boldsymbol{\Sigma}_{yx}$ is the cross-covariance matrix between $\boldsymbol{x}$ and $\boldsymbol{y}$.

Gaussian Distribution III

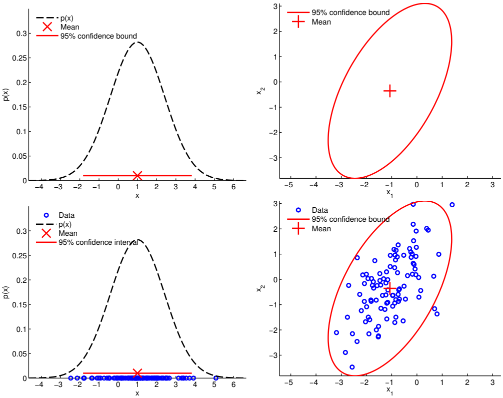


Figure: Gaussian distributions. Left column: Univariate (1-dimensional) Gaussians; Right column: Multivariate (2-dimensional) Gaussians, viewed from top. Crosses show the mean of the distribution, the red solid (contour) lines represent the 95% confident bounds. The bottom row

Gaussian Distribution IV

shows the Gaussians overlaid with 100 samples. We expect that on average 95/100 samples are within the red contour lines/intervals that indicate the 95% confidence bounds.

Gaussian Policies

- In robotics and other continuous control systems it makes sense to describe policies $\pi(a|s)$ as Gaussian distributions

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

$$\text{example: } \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t);\Sigma)$$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$

Policy gradients is trial-and-arror

We can distinguish likelihood based supervised learning (fitting states and actions to each other) from the reward-weighted fitting in policy gradient based reinforcement learning.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T}\nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i) \qquad \text{maximum likelihood:} \quad \nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$$

- The averages can be reinterpreted as weighted average over policies
- We increase the weight of trajectories that are good in reward
- We decrease the weight of trajectories that are bad in rewards

Section overview

Actor-Critic Methods

Learn, Plan, Act

Two uses of experience:

- model learning: to improve the model
- direct RL: directly improve the value function and policy

Improving value function and/or policy via a model is sometimes called indirect RL or model-based RL (or just "planning")
Two uses of experience:

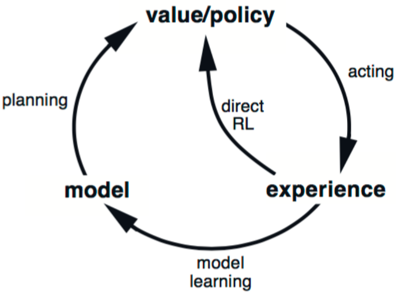
- Indirect (model-based) methods: get "better" policy with fewer interactions
- direct methods: simpler to implement and not affected by models

These two flavours can be meaningfully combined and can occur simultaneously and in parallel.

Actors & Critics

- Policy-Based methods (actors)
  - Find the optimal policy directly without the Q/V-function.
  - Policy-based are considered to have a faster convergence and are better for continuous and stochastic environments.
  - Examples: Policy-Based algorithms like Policy Gradients and REINFORCE
- Value Based methods (critics)
  - Find/approximate optimal value function (mapping action to value).
  - These are considered more sample efficient and steady.
  - Examples: Q Learning, Deep Q Networks, Double Dueling Q Networks, etc

Actor-Critic Methods I



- The principal idea is to split the RL model in two: one for computing an action based on a state and another one to produce the Q values of the action.