Lecture 6: Policy Gradients and Actor Critics

Hado van Hasselt

UCL 2018

Vapnik's rule

"Never solve a more general problem as an intermediate step."

— Vladimir Vapnik, 1998

If we care about optimal behaviour: why not learn a policy directly?

General overview

- Model-based RL:
  - \+ ‘Easy’ to learn a model (supervised learning)
  - \+ Learns ‘all there is to know’ from the data
  - \- Objective captures irrelevant information
  - \- May focus compute/capacity on irrelevant details
  - \- Computing policy (planning) is non-trivial and can be computationally expensive
- Value-based RL:
  - \+ Closer to true objective
  - \+ Fairly well-understood — somewhat similar to regression
  - \- Still not the true objective — may still focus capacity on less-important details
- Policy-based RL:
  - \+ Right objective!
  - \- Ignores other learnable knowledge (potentially not the most efficient use of data)

Policy-Based Reinforcement Learning

- Previously we approximated paramteric value functions

$$v_{\mathbf{w}}(s) \approx v_\pi(s)$$
$$q_{\mathbf{w}}(s,a) \approx q_\pi(s,a)$$

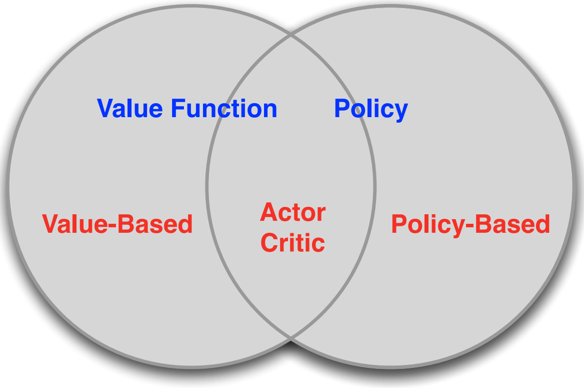
- A policy can be generated from these values
  - e.g., greedy, or ϵ-greedy
- In this lecture we will directly parametrize the policy directly

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

- We focus on model-free reinforcement learning

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy
    (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Advantages of Policy-Based RL

Advantages:

- Good convergence properties
- Easily extended to high-dimensional or continuous action spaces
- Can learn stochastic policies
- Sometimes policies are simple while values and models are complex
  - E.g., rich domain, but optimal is always go left

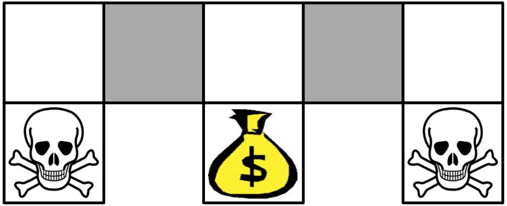
Disadvantages:

- Susceptible to local optima (especially with non-linear FA)
- Obtained knowledge is specific, does not always generalize well
- Ignores a lot of information in the data (when used in isolation)

Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)

Example: Aliased Gridworld (1)

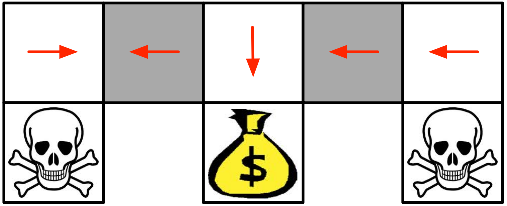


- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s,a) = (\overbrace{\underbrace{1}_{\text{N}}\ \underbrace{0}_{\text{E}}\ \underbrace{1}_{\text{S}}\ \underbrace{0}_{\text{W}}}^{\text{walls}}\ \overbrace{\underbrace{0}_{\text{N}}\ \underbrace{1}_{\text{E}}\ \underbrace{0}_{\text{S}}\ \underbrace{0}_{\text{W}}}^{\text{actions}})$$

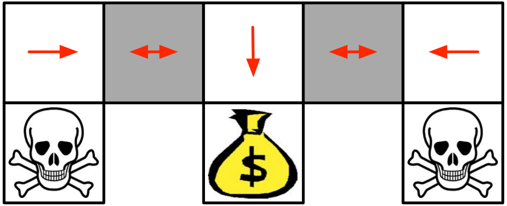
- Compare deterministic and stochastic policies

Example: Aliased Gridworld (2)



- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and never reach the money
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)



- An optimal stochastic policy moves randomly E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- Will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = v_{\pi_\theta}(s_1)$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s \mu_{\pi_\theta}(s) v_{\pi_\theta}(s)$$

  where $\mu_\pi(s) = p(S_t = s \mid \pi)$ is the probability of being in state s in the long run
  Think of is as the ratio of time spent in s under policy π
- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s \mu_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \sum_r p(r \mid s, a) r$$

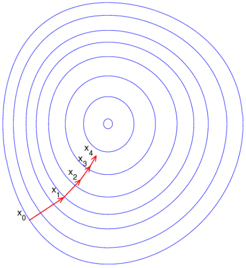
Policy Optimisation

- Policy based reinforcement learning is an optimization problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- We will focus on stochastic gradient ascent, which is often quite efficient (and easy to use with deep nets)

Policy Gradient

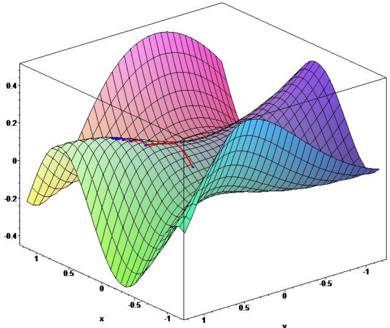
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$



- 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

Gradients on parameterized policies

- We need to compute an estimate of the policy gradient
- Assume policy π_θ is differentiable almost everywhere
  - E.g., π_θ is a linear function of the agent state, or a neural network
  - Or we could have a parameterized class of controllers
- Goal is to compute

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_d[v_{\pi_\theta}(S)] \,.$$

- We will use Monte Carlo samples to compute this gradient
- So, how does $\mathbb{E}_d[v_{\pi_\theta}(S)]$ depend on θ?

Contextual Bandits Policy Gradient

- Consider a one-step case (a contextual bandit) such that $J(\theta) = \mathbb{E}[R(S, A)]$. (Expectation is over μ (states) and π (actions))
- We cannot sample R_{t+1} and then take a gradient: R_{t+1} is just a number that does not depend on θ
- Instead, we use the identity:

$$\nabla_\theta \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_\theta \log \pi(A|S) R(S, A)] \, .$$

  (Proof on next slide)
- The right-hand side gives an expected gradient that can be sampled

The score function trick

$$
\begin{aligned}
\nabla_\theta \mathbb{E}[R(S, A)] &= \nabla_\theta \sum_s \mu(s) \sum_a \pi_\theta(a|s) R(s, a) \\
&= \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) R(s, a) \\
&= \sum_s \mu(s) \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} R(s, a) \\
&= \sum_s \mu(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) R(s, a) \\
&= \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S, A)]
\end{aligned}
$$

Contextual Bandit Policy Gradient

$$\nabla_\theta \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S, A)]$$ (see previous slide)

- This is something we can sample
- Our stochastic policy-gradient update is then

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t) \,.$$

- In expectation, this is the following the actual gradient
- So this is a pure stochastic gradient algorithm
- Intuition: increase probability for actions with high rewards

Example: Softmax Policy

- Consider a softmax policy on action preferences $h(s, a)$ as an example
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(a|s) = \frac{e^{h(s,a)}}{\sum_b e^{h(s,b)}}$$

- The gradient of the log probability is

$$\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta h(s, a) - \sum_b \pi_\theta(b|s) \nabla_\theta h(s, b)$$

Policy Gradient Theorem

- The policy gradient approach also applies to (multi-step) MDPs
- Replaces instantaneous reward R with long-term value $q_\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}\left[q_{\pi_\theta}(S, A) \nabla_\theta \log \pi_\theta(A|S)\right]$$

Expectation is over both states and actions

Policy gradients on trajectories

- Policy gradients do not need to know the dynamics
- Kind of surprising; shouldn't we know how the policy influences the states?

Policy gradients on trajectories: derivation

- Consider trajectory $\zeta = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \ldots$ with return $G(\zeta)$

$$\nabla_\theta J_\theta(\pi) = \nabla_\theta \mathbb{E}\left[G(\zeta)\right] = \mathbb{E}\left[G(\zeta) \nabla_\theta \log p(\zeta)\right] \qquad \text{(score function trick)}$$

$$\begin{aligned}
&\nabla_\theta \log p(\zeta) \\
&= \nabla_\theta \log \Big[p(S_0)\pi(A_0|S_0)p(S_1|S_0, A_0)\pi(A_1|S_1)\cdots \Big] \\
&= \nabla_\theta \Big[\log p(S_0) + \log \pi(A_0|S_0) + \log p(S_1|S_0, A_0) + \log \pi(A_1|S_1) + \cdots \Big] \\
&= \nabla_\theta \Big[\log \pi(A_0|S_0) + \log \pi(A_1|S_1) + \cdots \Big]
\end{aligned}$$

So:

$$\nabla_\theta J_\theta(\pi) = \mathbb{E}\left[G(\zeta) \nabla_\theta \sum_{t=0} \log \pi(A_t|S_t) \right] = \mathbb{E}\left[\left(\sum_{t=0} R_{t+1} \right) \left(\nabla_\theta \sum_{t=0} \log \pi(A_t|S_t) \right) \right]$$

Policy gradients on trajectories: reduce variance

- Note that, in general

$$
\begin{aligned}
\mathbb{E}\left[b\nabla_\theta \log \pi(A_t|S_t)\right] &= \mathbb{E}\left[\sum_a \pi(a|S_t) b \nabla_\theta \log \pi(a|S_t)\right] \\
&= \mathbb{E}\left[b\nabla_\theta \sum_a \pi(a|S_t)\right] \\
&= \mathbb{E}\left[b\nabla_\theta 1\right] \\
&= 0
\end{aligned}
$$

- This holds only if b does not depend on the action (though it can depend on the state)
- Implies we can subtract a baseline to reduce variance

Policy gradients on trajectories: reduce variance

- Consider trajectory $\zeta = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \ldots$ with return $G(\zeta)$

$$\nabla_\theta J_\theta(\pi) = \mathbb{E}\left[\left(\sum_{t=0} R_{t+1}\right)\left(\nabla_\theta \sum_{t=0} \log \pi(A_t|S_t)\right)\right]$$

but $\sum_{t=0}^{k} R_{t+1}$ does not depend on actions $A_{k+1}, A_{k+2}, \ldots$, so

$$= \mathbb{E}\left[\sum_{t=0} \nabla_\theta \log \pi(A_t|S_t) \sum_{i=0} R_{i+1}\right] \qquad \text{(rewrite of above)}$$

$$= \mathbb{E}\left[\sum_{t=0} \nabla_\theta \log \pi(A_t|S_t) \sum_{i=t} R_{i+1}\right]$$

$$= \mathbb{E}\left[\sum_{t=0} \nabla_\theta \log \pi(A_t|S_t) q_\pi(S_t, A_t)\right]$$

Policy gradients on trajectories: reduce variance

- A good baseline is $v_\pi(S_t)$

$$\nabla_\theta J_\theta(\pi) = \mathbb{E}\left[\sum_{t=0} \nabla_\theta \log \pi(A_t|S_t)(q_\pi(S_t, A_t) - v_\pi(S_t))\right]$$

- Typically, we estimate $v_{\boldsymbol{w}}(s)$ explicitly, and sample

$$q_\pi(S_t, A_t) \approx G_t^{(n)}$$

- For instance, $G_t^{(1)} = R_{t+1} + \gamma v_{\boldsymbol{w}}(S_{t+1})$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- What is the value of policy π_θ for current parameters θ?
- This problem was explored in previous lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - *n*-step TD

Actor-Critic

Critic Update parameters $\boldsymbol{w}$ of $v_{\boldsymbol{w}}$ by n-step TD (e.g., $n = 1$)

Actor Update θ by policy gradient

function ADVANTAGE ACTOR CRITIC
  Initialise s, θ
  for t = 0, 1, 2, ... **do**
    Sample $A_t \sim \pi_\theta(S_t)$
    Sample R_{t+1} and S_{t+1}
    $\delta_t = R_{t+1} + \gamma v_{\boldsymbol{w}}(S_{t+1}) - v_{\boldsymbol{w}}(S_t)$ [one-step TD-error, or advantage]
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta\ \delta_t\ \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(S_t)$ [TD(0)]
    $\theta \leftarrow \theta + \alpha\ \delta_t\ \nabla_\theta \log \pi_\theta(A_t \mid S_t)$ [Policy gradient update]
  end for
end function

Full advantage actor critic agent

- Advantage actor critic includes:
  - A representation (e.g., LSTM): $(S_{t-1}, O_t) \mapsto S_t$
  - A network $v_{\boldsymbol{w}}$: $S \mapsto v$
  - A network π_θ: $S \mapsto \pi$
  - Copies/variants π^m of π_θ to use as policies: $S_t^m \mapsto A_t^m$
  - A n-step TD loss on $v_{\boldsymbol{w}}$

    $$I(\boldsymbol{w}) = \frac{1}{2}\left(G_t^{(n)} - v_{\boldsymbol{w}}(S_t)\right)^2$$

    where $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} v_{\boldsymbol{w}}(S_{t+n})$
  - A n-step REINFORCE 'loss' on π_θ

    $$I(\theta) = \left[G_t^{(n)} - v_{\boldsymbol{w}}(S_t)\right] \log \pi_\theta(A_t|S_t)$$

  - Optimizers to minimize the losses
- Also know as A2C, or A3C (when combined with asynchronous parameter updates)

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
- Full returns: high variance
- One-step TD-error: high bias
- n-step TD-error: useful middle ground

$$\begin{aligned}\delta_t^{(n)} &= G_t^{(n)} - v_{\mathbf{w}}(S_t) \\ &= \underbrace{R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_{\mathbf{w}}(S_{t+n})}_{=G_t^{(n)}} - v_{\mathbf{w}}(S_t)\,.\end{aligned}$$

Bias in Actor-Critic Algorithms

- It is really important to use close-to on-policy targets
- If needed, use importance sampling to correct

$$G_t^{(n),\rho} = \frac{\pi_\theta(A_t \mid S_t)}{b(A_t \mid S_t)} \left(R_{t+1} + \gamma G_{t+1}^{(n-1),\rho} \right)$$

with

$$G_t^{(0),\rho} = v_{\boldsymbol{w}}(S_t) \approx v_\pi(S_t) \,.$$

λ-returns

- We can write a multi-step return recursively

$$G_t^{(n)} = R_{t+1} + \gamma G_{t+1}^{(n-1)}$$
$$G_t^{(0)} = v_{\mathbf{w}}(S_t) \approx v_\pi(S_t)\,.$$

- This is equivalent to

$$G_t^\lambda = R_{t+1} + \gamma(1-\lambda_{t+1})v_{\mathbf{w}}(S_{t+1}) + \gamma\lambda_{t+1}G_{t+1}^\lambda$$

with $\lambda_k = 1$ for $k \in \{t+1, \ldots, t+n-1\}$, and $\lambda_k = 0$ for $k = t+n$

- We can generalize to $\lambda_t \in [0, 1]$; this is called a λ-return
- It can be interpreted as a mixture of n-step returns
- One way to correct for off-policy returns: bootstrap (set $\lambda = 0$) whenever the policies differ
- Can be used for policy-gradient and value prediction

Trust region policy optimization

- Many extensions and variants exist
- Important: be careful with updates: a bad policy leads to bad data
- This is different from supervised learning (where learning and data are independent)
- One solution: regularise policy to not change too much

Increasing robustness with trust regions

- One way to prevent instability is to regularise
- A popular method is to limit the difference between subsequent policies
- For instance, use the Kullbeck-Leibler divergence:

$$\mathrm{KL}(\pi_{\text{old}} \| \pi_\theta) = \mathbb{E}\left[\int \pi_{\text{old}}(a \mid S) \log \frac{\pi_\theta(a \mid S)}{\pi_{\text{old}}(a \mid S)} \, \mathrm{d}a\right] .$$

  (a divergence is like a distance — but between distributions)
- Then maximise $J(\theta) - \eta \mathrm{KL}(\pi_{\text{old}} \| \pi_\theta)$, for some small η
- It can also help to use large batches

c.f. TRPO (Schulman et al. 2015) and PPO (Abbeel & Schulman 2016)

PPO: video

Continuous actions

- Because we direct update the policy parameters of the policy,
  we can easily deal with continuous action spaces
- Most algorithms discussed today can be used for discrete and continuous actions
- Exploration in high-dimensional continuous spaces can be challenging

Gaussian Policy

- In continuous action spaces, a Gaussian policy is common
- E.g., mean is some function of state $\mu(s)$
- For simplicity, lets consider fixed variance of σ^2
  (can be parametrized as well, instead)
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The gradient of the log of the policy is then

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{a - \mu(s)}{\sigma^2} \nabla \mu(s)$$

- This can be used, for instance, in REINFORCE / advantage actor critic

Continuous actor-critic learning automaton (Cacla)

- $a_t = \text{Actor}_\theta(S_t)$ (get current (continuous) action proposal)
- $A_t \sim \pi(\cdot|S_t, a_t)$ (e.g., $A_t \sim \mathcal{N}(a_t, \Sigma)$) (explore)
- $\delta_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$ (compute TD error)
- Update $v_{\mathbf{w}}(S_t)$ (e.g., using TD) (policy evaluation)
- If $\delta_t > 0$, update $\text{Actor}_\theta(S_t)$ towards A_t (policy improvement)
- If $\delta_t \leq 0$, do not update Actor_θ

Cacla: video

Gradient ascent on value

- REINFORCE works well in practice, but does not strongly exploit the critic
- If values generalize well, perhaps we can rely on them more
- Recall, the idea is to perform policy improvement
- Idea:
  1. Estimate $q_{\mathbf{w}} \approx q_\pi$, e.g., with Sarsa
  2. Configure actor, e.g., deterministic: $A_t = \pi_\theta(S_t)$
  3. Improve actor by gradient ascent:

$$\Delta\theta \propto \frac{\partial Q_\pi(s,a)}{\partial\theta} = \frac{\partial Q_\pi(s, \pi_\theta(S_t))}{\partial \pi_\theta(S_t)} \frac{\partial \pi_\theta(S_t)}{\partial\theta}$$

- Known under various names:
  "action-dependent heuristic dynamic programming" (ADHDP; Werbos 1990, Prokhorov & Wunsch 1997)
  "Deterministic policy gradient" (DPG; Silver et al. 2014)
  "Gradient ascent on the value" (GAV; van Hasselt & Wiering 2007)
- It's a form of policy iteration

Summary of Policy Gradient Algorithms

- The policy gradient has many forms

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(A|S)\ G_t\right] \qquad \text{REINFORCE}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(A|S)\ (G_t - b(S_t))\right] \qquad \text{REINFORCE}$$

$$\nabla_\theta J(\theta) \approx \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(A|S)\ \delta_t^{(n)}\right] \qquad \text{Advantage Actor-Critic}$$

$$\nabla_\theta J(\theta) \approx \nabla_\theta q_t(S, \pi_\theta(S)) \qquad \text{DPG}$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD) to estimate q_π or $v_\pi(s)$

Exploration

- The policy-gradient objective only considers improvement under current data
- Easy to get stuck in local optima — we need to explore
- We could use ϵ-greedy (assuming bounded action space), but that is not ideal
  - Wildly different actions may cause breakage
  - Exploration is mostly uninformed about current best guess
- Popular alternative: make sure entropy of the policy is not too low

Exploration and entropy

- The entropy of a policy is

$$-\sum_s \mu(s) \sum_a \pi(a \mid s) \log \pi(a \mid s) = -\mathbb{E}\left[\log \pi(A_t \mid S_t)\right]$$

- Idea: add a regularisation term that pushes up entropy slightly on each step
- Encourages exploration, but does not pick fully randomly
- E.g., may increase variance in Gaussian policies
- E.g., makes softmax slightly more uniform
- Somewhat similar to KL regularisation discussed before, but now regularising towards uniformly random policies
- Not a full solution to exploration, but works well in practice