# Lecture 6: Policy Gradient

Hado van Hasselt

# Outline

# 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)
  - Objective captures irrelevant information
  - Non-trivial going from model to policy (planning)
- Value-based RL:
  + Closer to true objective
  - Objective captures irrelevant information
- Policy-based RL:
  + Right objective!
  - Ignores other learnable knowledge
    (might be slow, might be overly specific)

# Policy-Based Reinforcement Learning

- Previously we approximated paramteric value functions

$$v_\theta(s) \approx v_\pi(s)$$
$$q_\theta(s, a) \approx q_\pi(s, a)$$

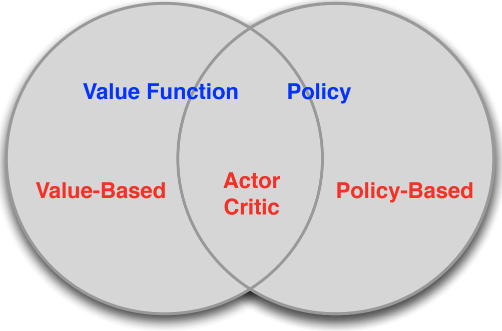
- A policy can be generated from these values
  - e.g., greedy, or $\epsilon$-greedy
- In this lecture we will directly parametrize the policy

$$\pi_\theta(a|s) = \mathbb{P}\left[a|s, \theta\right]$$

- We focus on model-free reinforcement learning

# Value-Based and Policy-Based RL

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



# Advantages of Policy-Based RL

Advantages:

- Better 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*

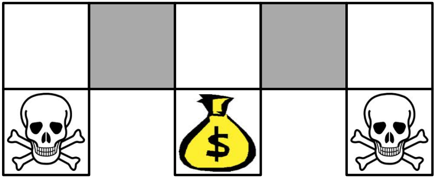
Disadvantages:

- Susceptible to local optima
- Obtained knowledge is specific, does not generalize

# 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)

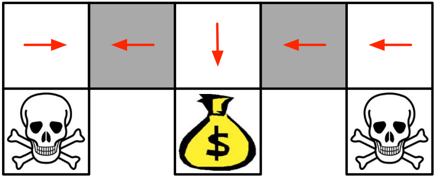


- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

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

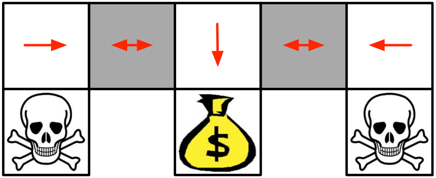
- Compare deterministic and stochastic policies

# 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 $\theta$, find best $\theta$
- But how do we measure the quality of a policy $\pi_\theta$?
- 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

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

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d_{\pi_\theta}(s)$ is stationary distribution of Markov chain for $\pi_\theta$

# Policy Optimisation

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

# Policy Gradient

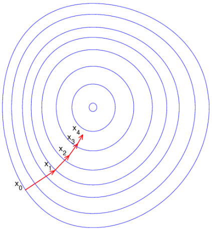
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters $\theta$

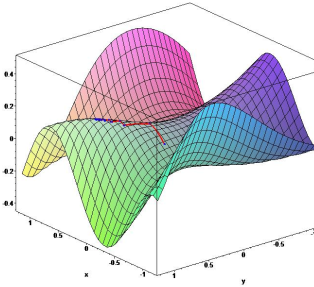
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- 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 $\alpha$ is a step-size parameter



# Gradients on parameterized policies

- We need to compute an estimate of the policy gradient
- Assume policy $\pi_\theta$ is differentiable almost everywhere
  - E.g., $\pi_\theta$ 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 $\theta$?

# Contextual Bandits Policy Gradient

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

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

# Contextual Bandit Policy Gradient

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

- 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 $x(s, a)$ as an example
- Probability of action is proportional to exponentiated weight

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

- The gradient of the log probability is

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

# Example: Bernoulli Policy

- Lets consider a bandit with two actions
- We play a game, and we win ($R_{t+1} = 1$) or lose ($R_{t+1} = -1$)
- Assume $\pi_\theta(a|s) = \frac{e^{x_a}}{e^{x_a} + e^{x_b}}$, where $\theta = \begin{bmatrix} x_a \\ x_b \end{bmatrix}$
- Recall $\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t)$
- Recall $\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta x(s, a) - \sum_b \pi_\theta(b|s) \nabla_\theta x(s, b)$
- Then $\nabla_\theta \log \pi_\theta(a) = (1 - \pi(a)) \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \pi(b) \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- So, if $x_a = x_b = 0$, we select action $a$, and we win:
  $\theta_{t+1} = \begin{bmatrix} \alpha(1 - \pi(a)) \\ -\alpha\pi(b) \end{bmatrix} = \begin{bmatrix} \alpha/2 \\ -\alpha/2 \end{bmatrix}$
- E.g., for $\alpha = 0.2$, policy goes from $\pi(a) = 0.5$ to $\pi(a) \approx 0.55$

# Policy Gradient Theorem

- REINFORCE can be applied to (multi-step) MDPs as well
- 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 $G(\zeta)$

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

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

So:

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

# Monte-Carlo Policy Gradient (REINFORCE)

- Using return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$ as an unbiased sample of $q_{\pi_\theta}(S_t, A_t)$
- Update parameters by stochastic gradient ascent

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(A_t|S_t) G_t$$

**function REINFORCE**
    Initialise $\theta$ arbitrarily
    **for** each episode $\{S_0, A_0, R_1, S_1 ..., S_{T-1}, A_{T-1}, R_T\} \sim \pi_\theta$ **do**
        $\theta \leftarrow \theta + \alpha \sum_{t=1}^{T-1} G_t \nabla_\theta \log \pi_\theta(A_t|S_t)$
    **end for**
**end function**

# Reducing variance with a baseline

- REINFORCE can have high variance
- One way to reduce variance is center use a state-dependent baseline $b(s)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(A_t|S_t)(G_t - b(s))$$

- This helps because it removes variance based on the value of the current state
- This works as long as the baseline does not depend on the policy parameters
- Good choice: $b(s) = v_\eta(s) \approx v_\pi(s)$ (learn with policy evaluation)

# Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- What is the value of policy $\pi_\theta$ for current parameters $\theta$?
- This problem was explored in previous lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD($\lambda$)

# Actor-Critic

- We can reduce variance further by bootstrapping

  Critic Update parameters $\eta$ of $v_\eta$ by TD
  Actor Update $\theta$ by policy gradient

**function** Advantage Actor Critic
  Initialise $s$, $\theta$
  Sample $A \sim \pi_\theta$
  **for** each step **do**
    Sample reward $R = \mathcal{R}_S^A$; sample transition $S' \sim \mathcal{P}_{S,\cdot}^A$
    Sample action $A' \sim \pi_\theta(S')$
    $\delta = R + \gamma v_\eta(S') - v_\eta(S)$ [TD-error, or advantage]
    $\eta \leftarrow \eta + \beta\ \delta\ \nabla_\eta v_\eta(S)$ [TD(0)]
    $\theta = \theta + \alpha\ \delta\ \nabla_\theta \log \pi_\theta(s, a)$ [Policy gradient update]
    $A \leftarrow A', S \leftarrow S'$
  **end for**
**end function**

# 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
- Can use $n$-step TD-error:

$$\delta_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_\eta(S_{t+n}) \,.$$

# Actor-Critic: video

# Continuous actions

- Because we operate on the parameters of the policy, we can easily deal with continuous action spaces
- Most algorithms discussed today can be used, almost out of the box, for both 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 $\mu(s)$
- Lets assume variance is fixed at $\sigma^2$
  (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

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

# Continuous actor-critic learning automaton (Cacla)

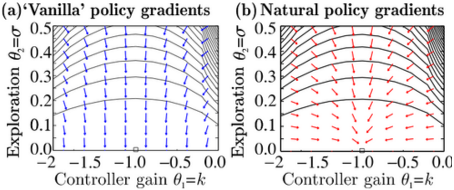
- $a_t = \text{Actor}_\theta(S_t)$ (get current (continuous) action proposal)
- $A_t \sim \pi(\cdot|S_t, a_t)$ (e.g., $A_t \sim \mathcal{N}(a_t, \Sigma)$) (explore)
- $\delta_t = R_{t+1} + \gamma v_\eta(S_{t+1}) - v_\eta(S_t)$ (compute TD error)
- Update $v_\eta(S_t)$ (e.g., using TD) (policy evaluation)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

# Cacla: video

# Alternative Policy Gradient Directions

- Gradient ascent algorithms can follow *any* ascent direction
- A good ascent direction can significantly speed convergence
- Also, a policy can often be reparametrised without changing action probabilities
- For example, increasing score of all actions in a softmax policy
- The vanilla gradient is sensitive to these reparametrisations

# Natural Policy Gradient




- The natural policy gradient is parametrisation independent
- It finds direction that maximally ascends objective function, when changing policy by a small, fixed amount

$$\nabla_\theta^{nat} \pi_\theta(s,a) = F_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

- where $F_\theta$ is the Fisher information matrix

$$F_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T\right]$$

# 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 $q_\eta \approx q_\pi$, 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 as DPG (Silver et al. 2014), ADHDP (Werbos 1990) or, simply, gradient ascent on value (van Hasselt & Wiering 2007)
- A form of policy iteration

# 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
- E.g., restrict Kullback-Leibler divergence: $KL(\pi_{t+1} \| \pi_t) < c$, for some small $c$ (Schulman et al. 2015)
- Intuition: if the policy does not change too much, the approximations remain more valid

# Summary of Policy Gradient Algorithms

- The policy gradient has many forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S)\ G_t\right] && \text{REINFORCE} \\
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S)\ (G_t - b(S_t))\right] && \text{REINFORCE} \\
\nabla_\theta J(\theta) &\approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S)\ \delta_t^{(n)}\right] && \text{Advantage Actor-Critic} \\
\nabla_\theta J(\theta) &\approx \nabla_\theta Q(S, \pi_\theta(S)) && \text{DPG}
\end{aligned}
$$

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