Policy Gradient & Actor-Critic Methods

Recap

Dan Beechey

djeb20@bath.ac.uk

Reinforcement Learning
Department of Computer Science
University of Bath

Outline

- Introduction to Policy-Based Methods.
- Deriving A Simple Policy Gradient
- The REINFORCE Algorithm
    - Reward-To-Go Policy Gradient
    - Baseline Functions
- Actor-Critic Methods
- The Deep Deterministic Policy Gradient (DDPG) Algorithm

Types of Reinforcement Learning Methods

- **Value-Based** RL
  - Approximates the **optimal action-value function** $Q^*(s, a)$.

Last Lecture

Deep RL uses deep neural networks to represent the value function

Types of Reinforcement Learning Methods

- **Value-Based** RL
  - Approximates the **optimal action-value function** $Q^*(s, a)$.
- **Policy-Based** RL
  - Directly search the policy-space for the **optimal policy** $\pi^*(a|s)$.

This Lecture

Deep RL uses deep neural networks to represent the value function or policy.

N.B. these are not mutually exclusive!

Why Policy-Based Methods?

- So far, we have worked with **value-based** methods.
  - We'd learn the action-value function, then derive a policy (e.g. ϵ-greedy).
- What if the optimal policy is **stochastic**?
  - Value-based methods have no natural way of dealing with this.


- Instead, could we learn the optimal policy directly?

Value-Based Deep RL Methods

$Q_{\boldsymbol{\theta}}(s, a)$ = Expected return from choosing action a in state s and following some policy π thereafter, given parameters $\boldsymbol{\theta}$.

- We can represent our action-value function as some differentiable function of state s and action a with parameters $\boldsymbol{\theta}$.
  - E.g. as a Neural Network.

- We can then optimise our Q-function via stochastic gradient descent.

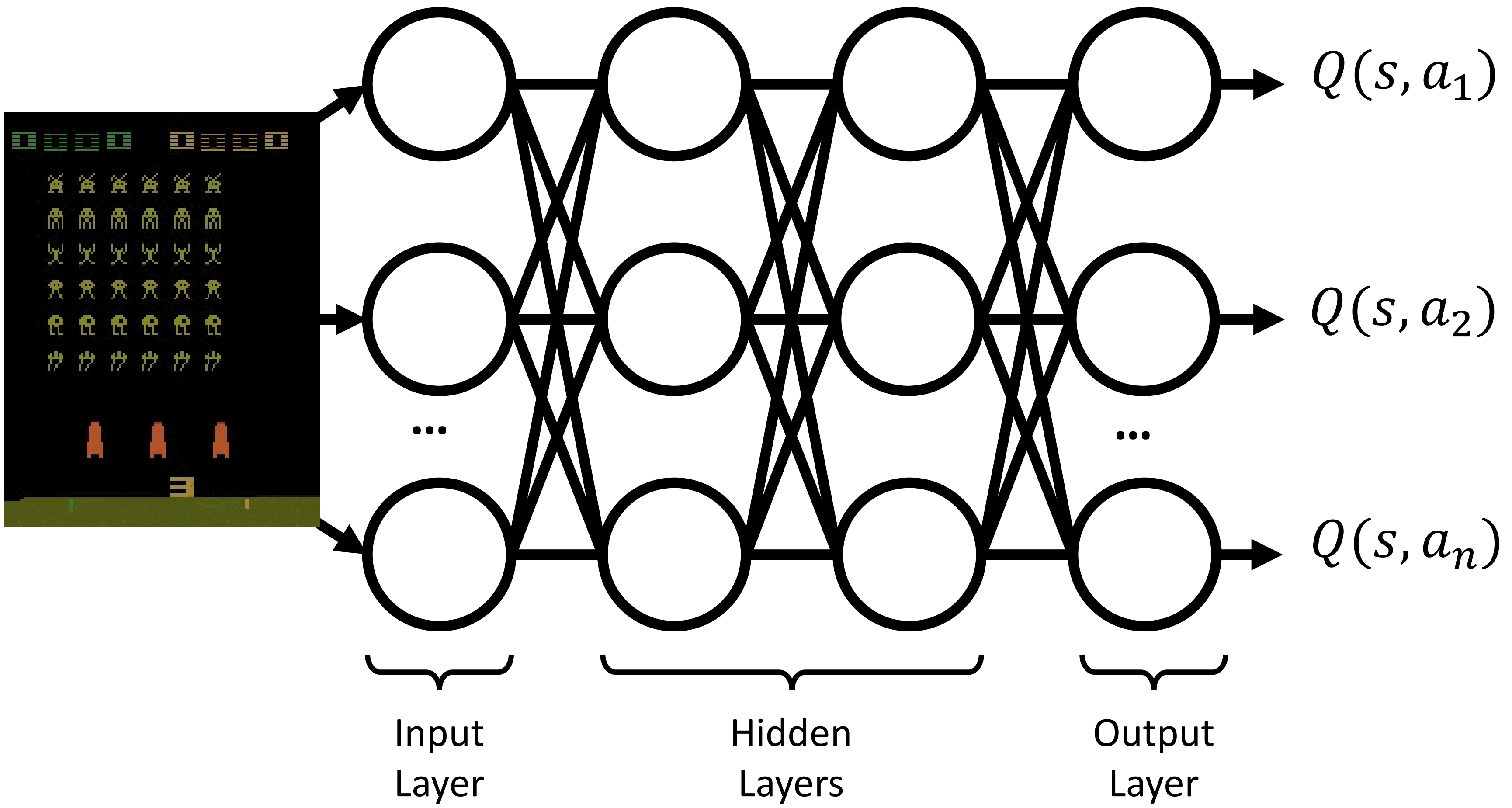
Policy-Based Deep RL Methods

$\pi_{\boldsymbol{\theta}}(a|s)$ = Probability of choosing action a given state s and policy parameters $\boldsymbol{\theta}$.

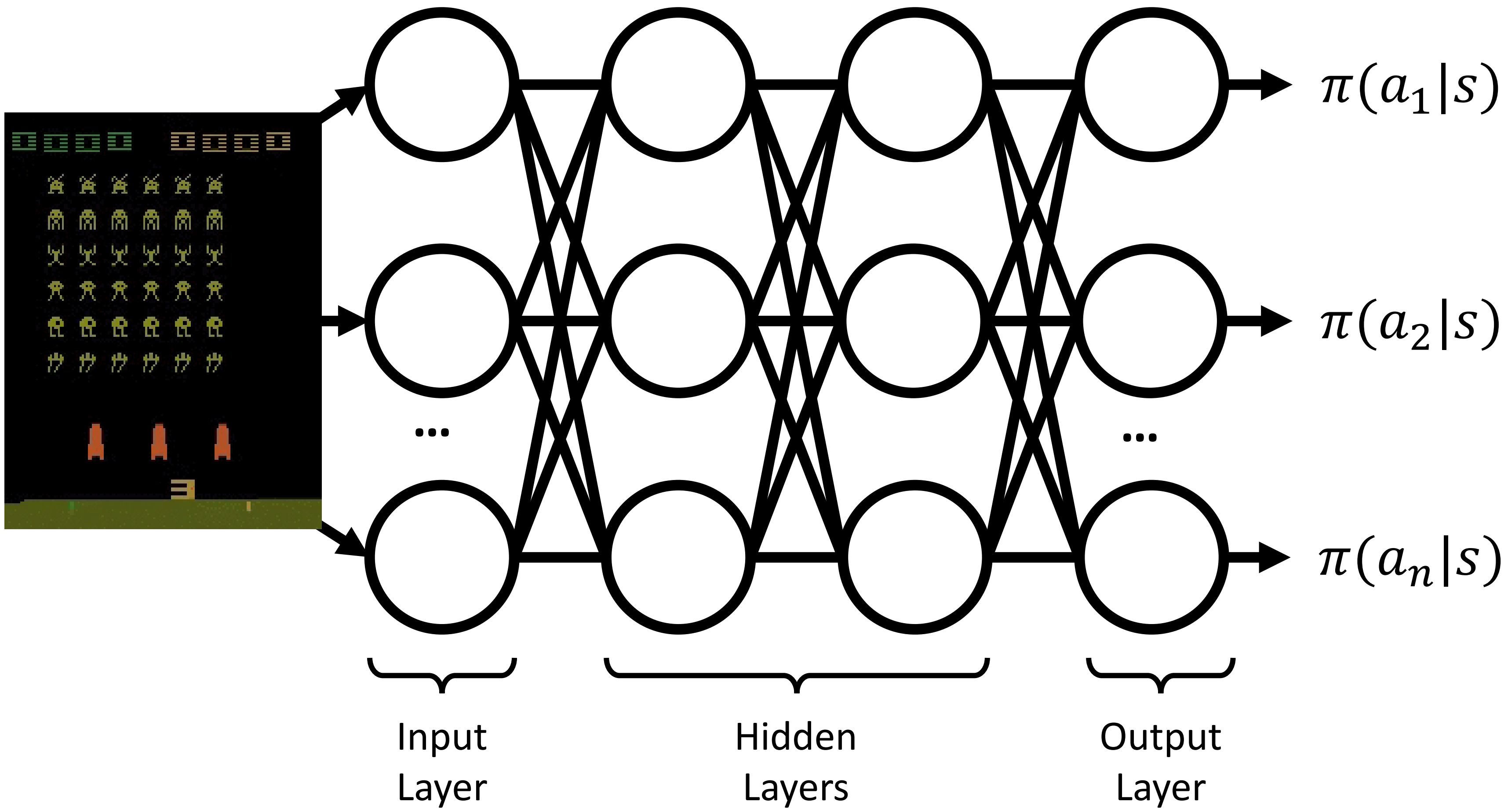
- We can represent the policy as some differentiable function of the state s with parameters $\boldsymbol{\theta}$.
  - E.g. as a Neural Network.

- We can then optimise the policy via stochastic gradient descent.

Q-Network

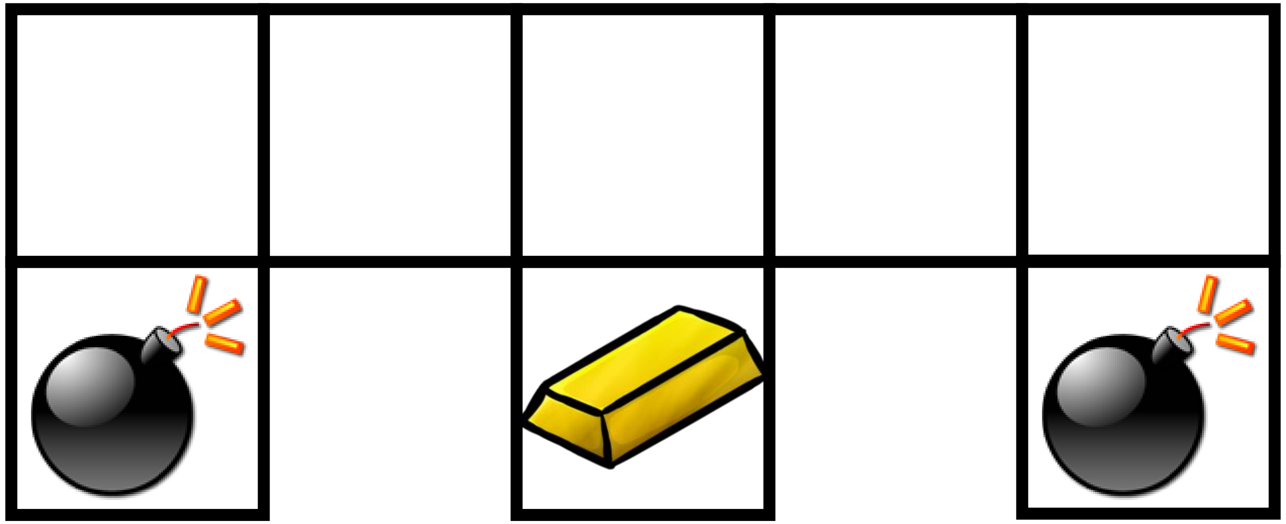


Policy Network



Aliased Gridworld

State = (Wall to North, Wall to South, Wall to East, Wall to West)

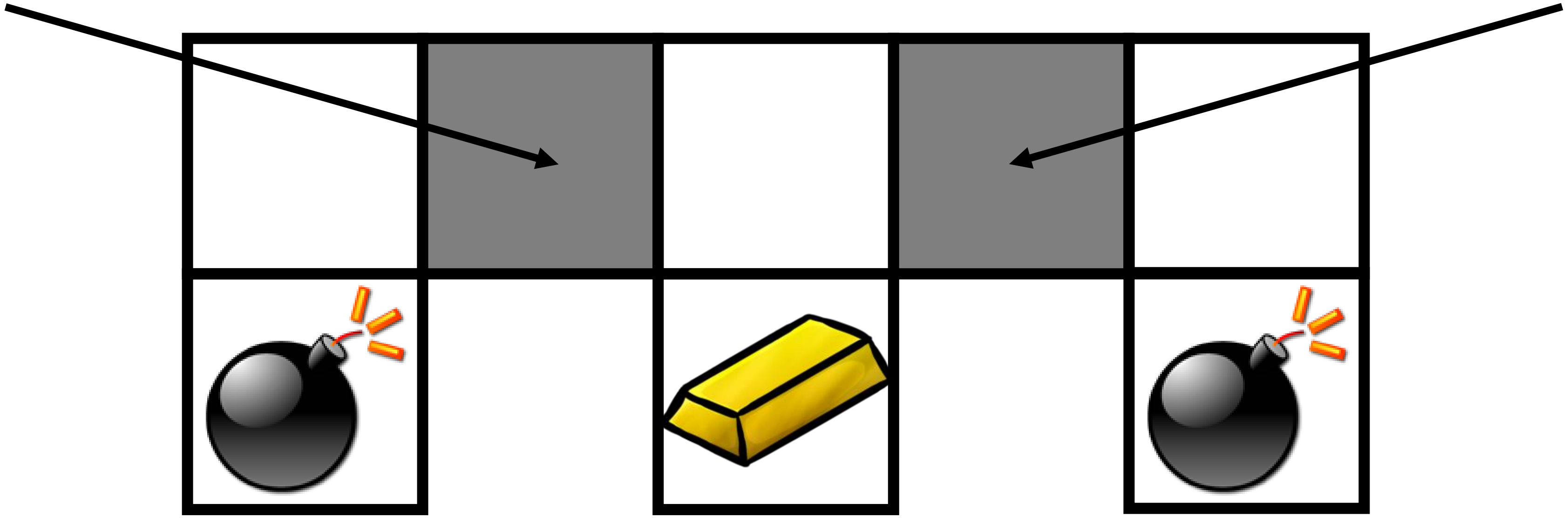


Example Credit: [David Silver]

Aliased Gridworld

State = (Wall to North, Wall to South, Wall to East, Wall to West)

(TRUE, TRUE, FALSE, FALSE) (TRUE, TRUE, FALSE, FALSE)

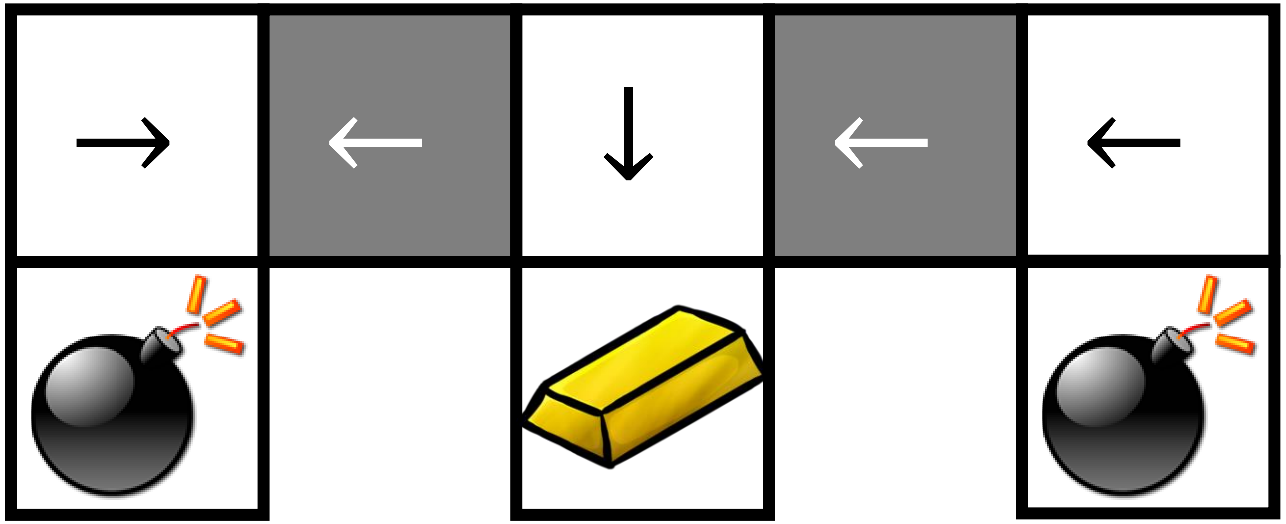


These two states have identical representations!

Example Credit: [David Silver]

Aliased Gridworld

State = (Wall to North, Wall to South, Wall to East, Wall to West)

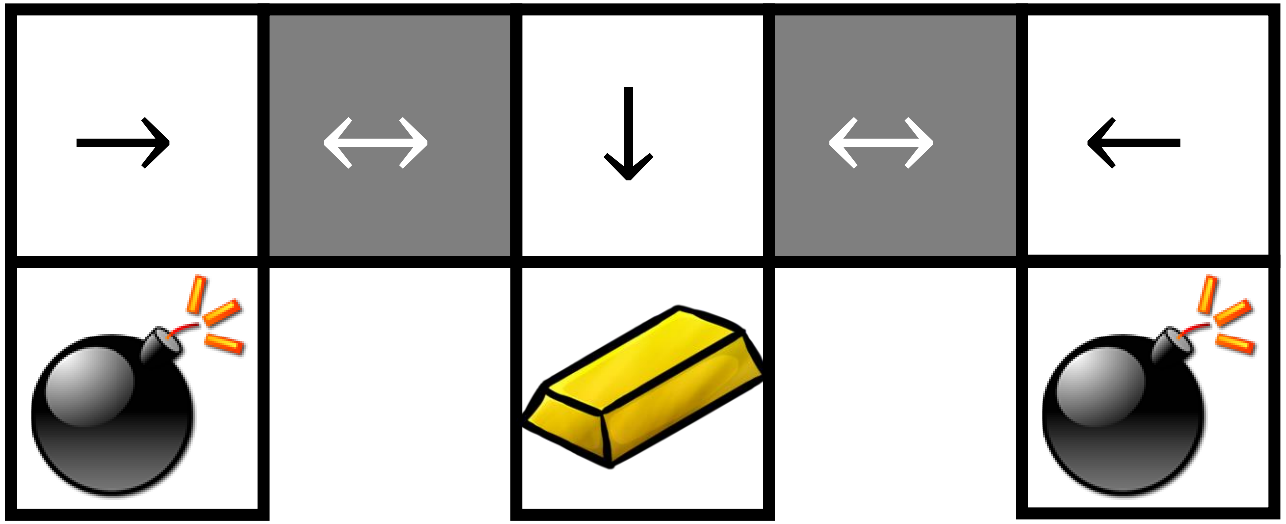


These two states have identical representations!
A deterministic policy would not work well here.

Example Credit: [David Silver]

Aliased Gridworld

State = (Wall to North, Wall to South, Wall to East, Wall to West)



These two states have identical representations!

A stochastic policy would work much better!

Example Credit: [David Silver]

Policy Gradient Methods

$\pi_{\boldsymbol{\theta}}(a|s)$ = Probability of choosing action a given state $\boldsymbol{s}$ and policy parameters $\boldsymbol{\theta}$.

Policy Gradient Methods

$\pi_{\boldsymbol{\theta}}(a|s)$ = Probability of choosing action a given state $\boldsymbol{s}$ and policy parameters $\boldsymbol{\theta}$.

- $J(\pi_{\boldsymbol{\theta}})$ is some **objective function** for our policy, which we aim to maximise (e.g. undiscounted expected return).

Policy Gradient Methods

$\pi_{\boldsymbol{\theta}}(a|s)$ = Probability of choosing action a given state $\boldsymbol{s}$ and policy parameters $\boldsymbol{\theta}$.

- $J(\pi_{\boldsymbol{\theta}})$ is some **objective function** for our policy, which we aim to maximise (e.g. undiscounted expected return).
- Policy gradient update rule: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J\left(\pi_{\boldsymbol{\theta}_t}\right)$

Policy Gradient Methods

$\pi_{\boldsymbol{\theta}}(a|s)$ = Probability of choosing action a given state $\boldsymbol{s}$ and policy parameters $\boldsymbol{\theta}$.

- $J(\pi_{\boldsymbol{\theta}})$ is some **objective function** for our policy, which we aim to maximise (e.g. undiscounted expected return).

- Policy gradient update rule: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J\left(\pi_{\boldsymbol{\theta}_t}\right)$

- $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ is called the **policy gradient**.

Policy Gradient Methods

- Policy gradient update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J\left(\pi_{\boldsymbol{\theta}_t}\right)$

- In order to use this update rule in an algorithm, we need an expression for the policy gradient which we can numerically compute.
- This expression should depend only on π, $\nabla_{\boldsymbol{\theta}} \pi$, $\boldsymbol{\theta}_{\boldsymbol{t}}$, and J.

- We will derive this expression for the policy gradient over the next few slides, and then use it later on in an actual learning algorithm.

A Few Definition Reminders…

- A state-action **trajectory** $\tau = (S_0, A_0, R_0, \ldots, S_{T-1}, A_{T-1}, R_{T-1}, S_T)$.

- The **return of a trajectory** $G(\tau) = \sum_{t=0}^{T} R_t$
  - We'll be using the undiscounted return, simply the sum of rewards.

- Our **objective function** $J(\pi_{\boldsymbol{\theta}})$ is the **expected return**:

$$J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G(\tau)]$$

- The **expected value** of a continuous random variable X with probability density function $p(x)$ is $\int_{-\infty}^{\infty} x \cdot P(x)$.

Deriving the Policy Gradient

- We need to find the gradient of our objective function.
  - Assume that we are using the undiscounted expected return as our objective function, and we have full access to π, $\log \pi$ and their derivatives.

- Let's derive this step-by-step together!

- We'll assume that we'll have access to $\pi_{\boldsymbol{\theta}}$, $\log \pi_{\boldsymbol{\theta}}$ and their derivatives.
  - When using a deep neural network, we will do (e.g. via backpropagation).

Deriving the Policy Gradient

- Step 1: What are we trying to find?

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G(\tau)]$$

Deriving the Policy Gradient

- Step 2: Expand Expectation

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G(\tau)]$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \int_{\tau} P(\tau|\boldsymbol{\theta}) G(\tau)$$

Deriving the Policy Gradient

- Step 3: Bring Gradient Under Integral

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \int_{\tau} P(\tau|\boldsymbol{\theta}) G(\tau)$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\tau} \nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta}) G(\tau)$$

Deriving the Policy Gradient

- Step 4: Log-Derivative Trick

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\tau} \nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta}) G(\tau)$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\tau} P(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})\, G(\tau)$$

Recall from calculus:

$$\frac{\partial}{\partial x} \log f(x) = \frac{f'(x)}{f(x)}$$

Deriving the Policy Gradient

- Step 5: Return to Expectation Form

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\tau} P(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})\, G(\tau)$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})\, G(\tau)]$$

Deriving the Policy Gradient

- Step 5: Return to Expectation Form

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\tau} P(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})\, G(\tau)$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[\underline{\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})}\, G(\tau)]$$

Deriving the Policy Gradient

Recall that:

$$\log \prod_{i=0}^{n} P(x_i) = \sum_{i=0}^{n} \log P(x_i)$$

- What is $\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})$?

$$P(\tau|\boldsymbol{\theta}) = P(S_0) \cdot \prod_{t=0}^{T} P(S_t, A_t, R_t, S_{t+1}) \cdot \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

$$\log P(\tau|\boldsymbol{\theta}) = \log P(S_0) + \sum_{t=0}^{T} \log P(S_t, A_t, R_t, S_{t+1}) + \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

$$\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log P(S_0) + \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log P(S_{t+1}, R_t|S_t, A_t) + \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

Deriving the Policy Gradient

Recall that:

$$\log \prod_{i=0}^{n} P(x_i) = \sum_{i=0}^{n} \log P(x_i)$$

- What is $\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})$?

$$P(\tau|\boldsymbol{\theta}) = P(S_0) \cdot \prod_{t=0}^{T} P(S_t, A_t, R_t, S_{t+1}) \cdot \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

$$\log P(\tau|\boldsymbol{\theta}) = \log P(S_0) + \sum_{t=0}^{T} \log P(S_t, A_t, R_t, S_{t+1}) + \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

$$\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta}) = \cancel{\nabla_{\boldsymbol{\theta}} \log P(S_0)} + \sum_{t=0}^{T} \cancel{\nabla_{\boldsymbol{\theta}} \log P(S_{t+1}, R_t|S_t, A_t)} + \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

Deriving the Policy Gradient

- Step 6: Substitute Grad-Log-Prob of Trajectory

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})\, G(\tau)]$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\, G(\tau)\right]$$

Deriving the Policy Gradient

- All Done!

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)\, G(\tau) \right]$$

We now have an expression for our policy-gradient which depends only on π, $\log \pi$, their derivatives.

Deriving the Policy Gradient

- All Done!

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) G(\tau)$$

Since our expression is an expectation, we can approximate it by taking the sample mean over many trajectories $\tau \in D$.

Given a trajectory τ

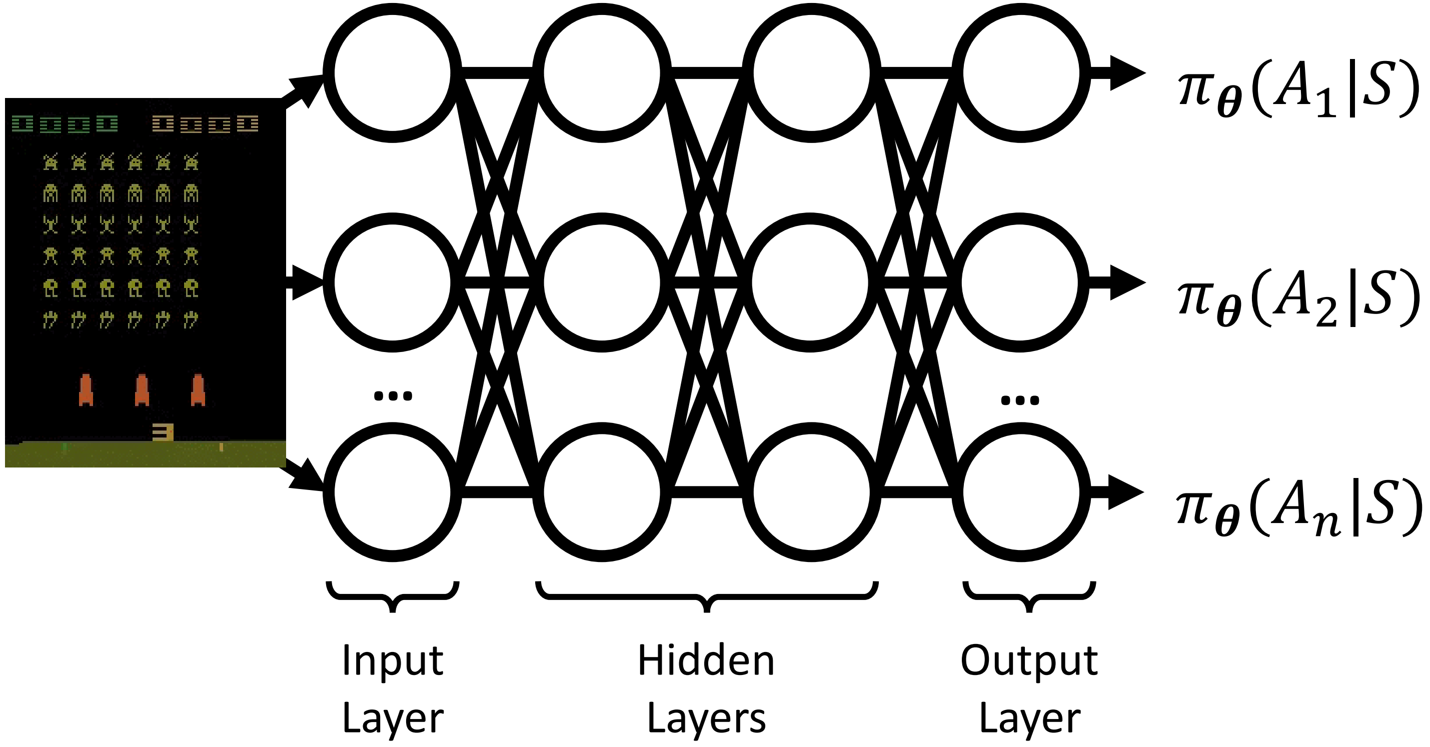
$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[R(\tau)] & \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) & \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) & \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) & \text{Log-derivative trick} \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)R(\tau)] & \text{Return to expectation form} \\
\therefore \nabla_\theta J(\pi_\theta) &= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] & \text{Expression for grad-log-prob}
\end{aligned}
$$

We can approximate this with the sample mean of many trajectories $\tau \in D$:

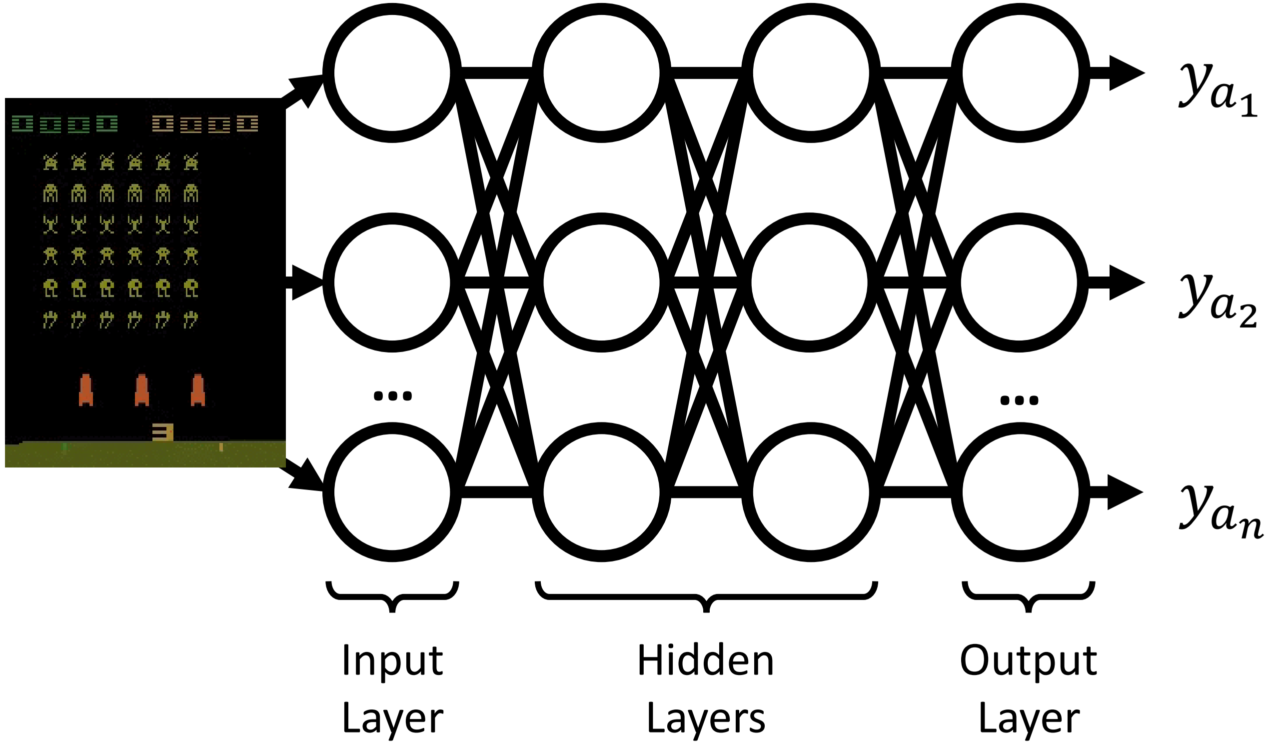
$$
\nabla_\theta J(\pi_\theta) \approx \frac{1}{|D|}\sum_{\tau\in D}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)
$$

Image Credit: [OpenAI]

Policy Network – Action Selection

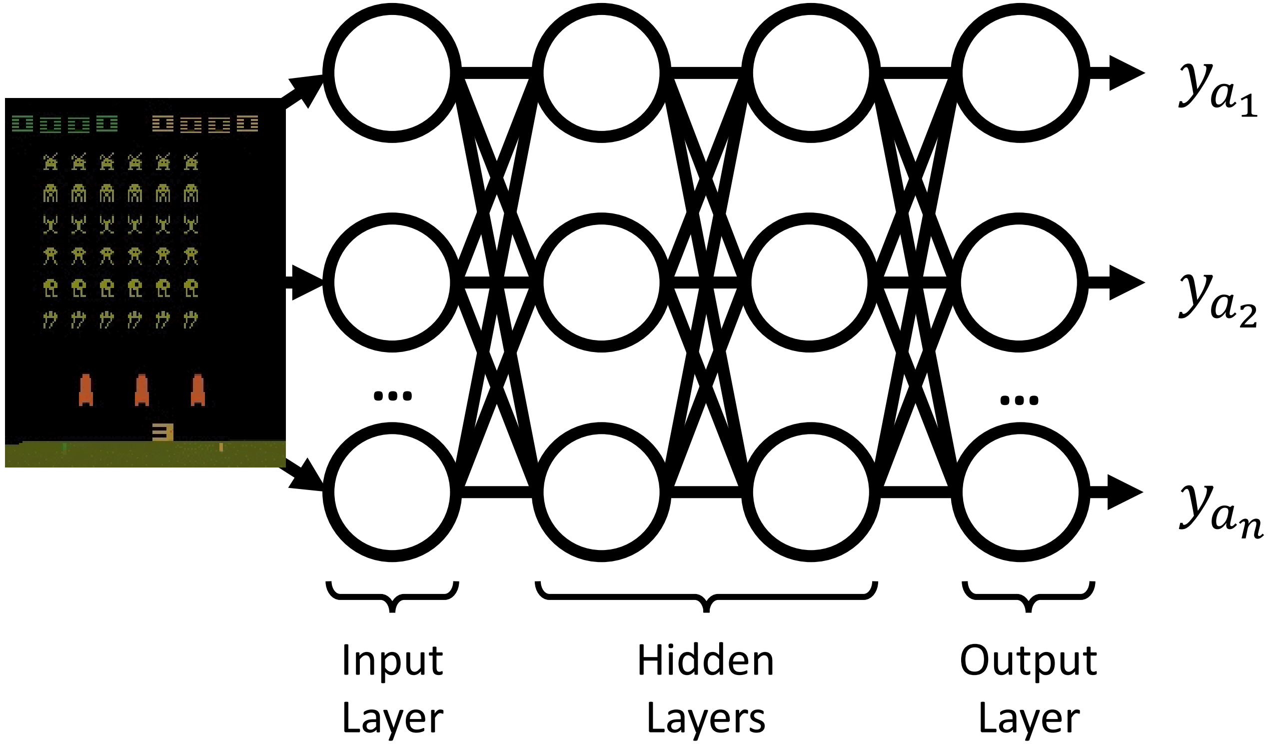


Policy Network – Action Selection



- Raw output of our network won't actually be probabilities!
  - They'll be **action preferences**.

Policy Network – Action Selection



- Raw output of our network won't actually be probabilities!
  - They'll be **action preferences**.
- We can use soft-max over our network's outputs to perform action selection.

$$\pi_{\boldsymbol{\theta}}(A_i|s) = \frac{e^{y_{A_i}}}{\sum_j e^{y_{A_j}}}$$

Algorithm: REINFORCE

Initialise parameters: step size $\alpha \in (0,1]$

Initialise policy network π with parameters $\boldsymbol{\theta}$

For episode $= 1, M$ **do**

  Generate an episode trajectory $\tau \sim \pi_{\boldsymbol{\theta}}$

  For $t = 1, T-1$ **do**

    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)\, G$

  End For

End For

Algorithm: REINFORCE
Initialise parameters: step size $\alpha \in (0,1]$
Initialise policy network π with parameters $\boldsymbol{\theta}$

For episode $= 1, M$ **do**
    Generate an episode trajectory $\tau \sim \pi_{\boldsymbol{\theta}}$
    For $t = 1, T-1$ **do**
        $G \leftarrow \sum_{k=t+1}^{T} R_k$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)\, G$
    End For
End For

$\gamma = 1$ (No Discounting)

Algorithm: REINFORCE

Initialise parameters: step size $\alpha \in (0,1]$

Initialise policy network π with parameters $\boldsymbol{\theta}$

For episode $= 1, M$ **do**

  Generate an episode trajectory $\tau \sim \pi_{\boldsymbol{\theta}}$

  For $t = 1, T-1$ **do**

    $G \leftarrow \sum_{k=t+1}^{T} R_k$ ← Sample Return

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \underbrace{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)\, G}_{\text{Our Policy-Gradient!}}$

  End For

End For

$\gamma = 1$ (No Discounting)

Algorithm: REINFORCE

Initialise parameters: step size $\alpha \in (0,1]$

Initialise policy network π with parameters $\boldsymbol{\theta}$

For episode $= 1, M$ **do**

  Generate an episode trajectory $\tau \sim \pi_{\boldsymbol{\theta}}$

  For $t = 1, T-1$ **do**

    $G \leftarrow \sum_{k=t+1}^{T} R_k$ ← Sample Return

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \underbrace{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\, G}_{\text{Our Policy-Gradient!}}$

  End For

End For

...or is it?

$\gamma = 1$ (No Discounting)

Reward-To-Go Policy Gradient

- Currently, we update the log-probabilities of each action in proportion to the sum of **all rewards taken during a trajectory**.

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)\, \underbrace{G(\tau)}_{\text{Sum of all rewards.}}\right]$$

- This doesn't make much sense. Instead, we could **only reinforce actions based on the rewards earned after they were executed**.

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \underbrace{\sum_{t'=t}^{T} r(S_{t'}, A_{t'}, S_{t'+1})}_{\text{Sum of all rewards after time-step } t.}\right]$$

Advantages of Policy-Gradient Methods

- Deals naturally with stochastic policies.

- Stochastic policies explore naturally.

- Stronger convergence guarantees than value-based methods.

Baseline Functions

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \left(\sum_{t'=t}^{T} r(S_{t'}, A_{t'}, S_{t'+1}) \right) \right]$$

Baseline Functions

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \left(\sum_{t'=t}^{T} r(S_{t'}, A_{t'}, S_{t'+1}) \right) \right]$$

- Currently, there will be **high variance in the magnitude of our updates**.
  - Trajectories may vary a lot between runs.
  - Some states will have a high value, others will have a low value.
- We can't avoid variance due to stochasticity in our policy or the environment.
- We can take into account variance in state values.

Baseline Functions

- We want to reduce the variance in the magnitude of our updates.
- We can reduce the variance of the updates by subtracting a **baseline function** $b(s_t)$.

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\left(\sum_{t'=t}^{T} r(S_{t'}, A_{t'}, S_{t'+1}) - b(S_t)\right)\right]$$

- We can use any function as a baseline, as long as it doesn't change our policy gradient in expectation.

Baseline Functions

- We want to reduce the variance in the magnitude of our updates.
- We can reduce the variance of the updates by subtracting a **baseline function** $b(s_t)$.

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathrm{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t) \left(\sum_{t'=t}^{T} r(S_{t'}, A_{t'}, S_{t'+1}) - V_\pi(S_t)\right)\right]$$

- We can use any function as a baseline, as long as it doesn't change our policy gradient in expectation.
- The value function $V_\pi(s_t)$ is a useful baseline function.

Combining Value & Policy-Based Methods

- Using baseline functions, we can combine value-based methods and policy-based methods!
  - We can use the state-value or action-value function as baselines!
- Instead of having one function representing either the value function or a policy, we'll need both!
  - If we use a state-value function or action-value function as part of our policy gradient updates, we will need to learn it separately.

Actor-Critic Methods

- Alongside using value functions as baselines, we can also use them to for bootstrapping. This lets us move away from Monte Carlo updates.

- This gives us all the previous benefits that we've seen from bootstrapping, and the benefits of policy-based methods.

- We call bootstrapping methods which learn the policy function directly using estimates from value functions **Actor-Critic Methods**.
  - The policy function is the "**Actor**".
  - The value function is the "**Critic**".

Types of Reinforcement Learning Methods

- **Value-Based** RL
  - Approximates the **optimal action-value function** $Q^*(s, a)$.
- **Policy-Based** RL
  - Directly search the policy-space for the **optimal policy** $\pi^*(a|s)$.

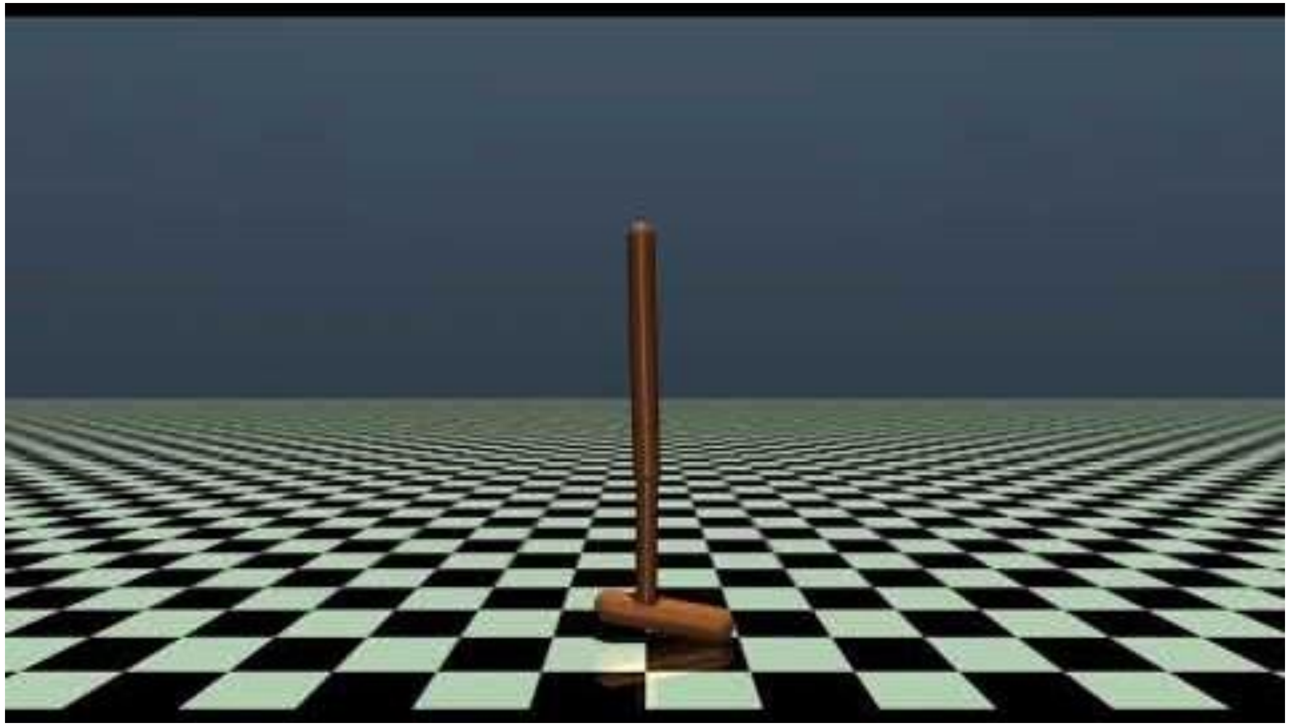
Actor-Critic methods do both!

Deep RL uses deep neural networks to represent the value function or policy

N.B. these are not mutually exclusive! (See, I told you!)

Deep Deterministic Policy Gradients

- DDPG is an actor-critic algorithm for continuous action-spaces.
- It makes use of many of DQN's tricks, such as replay buffers and target networks.
- It is off-policy, so can make use of old experiences (unlike REINFORCE).
- It makes policy-gradient updates maximising $\mathrm{E}[Q(s,a)]$.

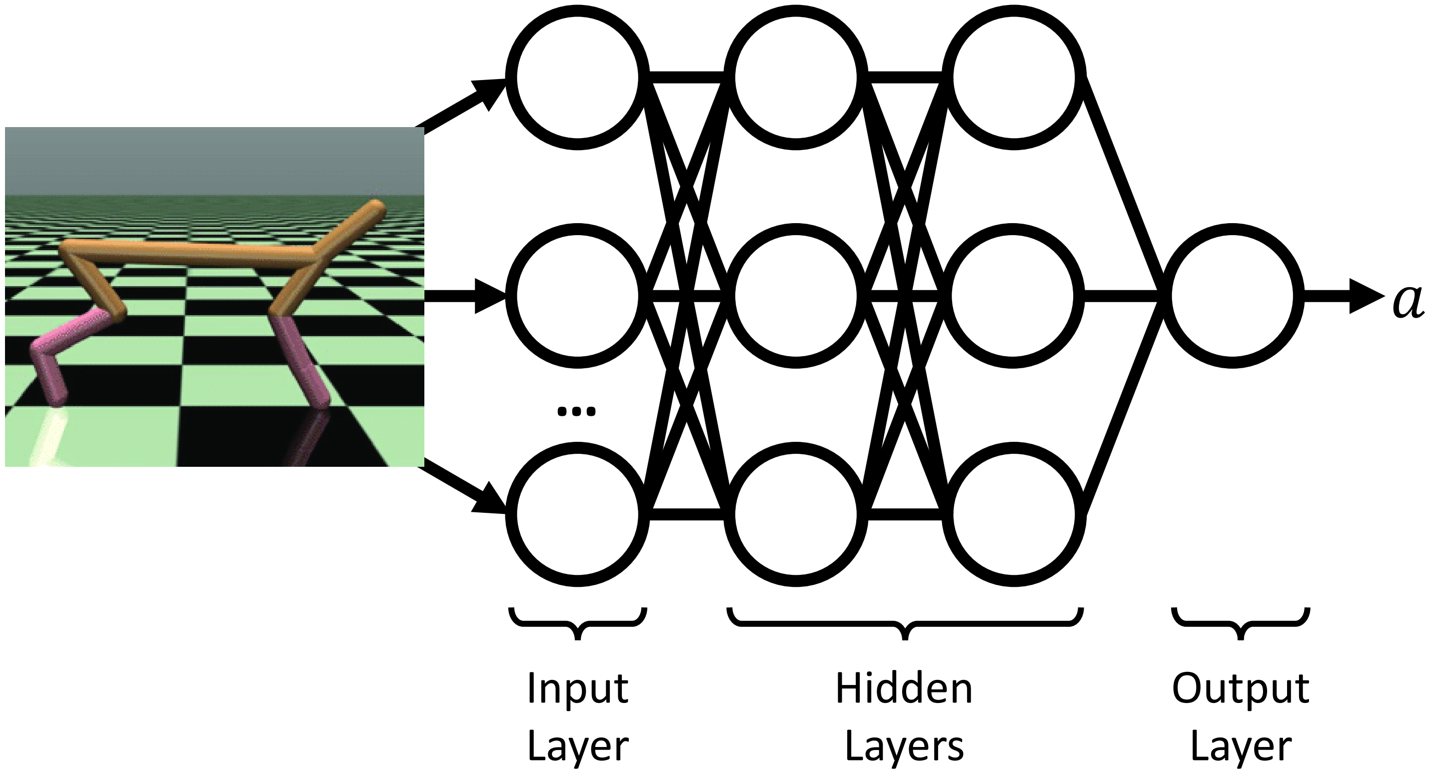


Recall: We maximised $\mathrm{E}[R(\tau)]$ earlier!

Video Credit: [Sam Kirkiles]

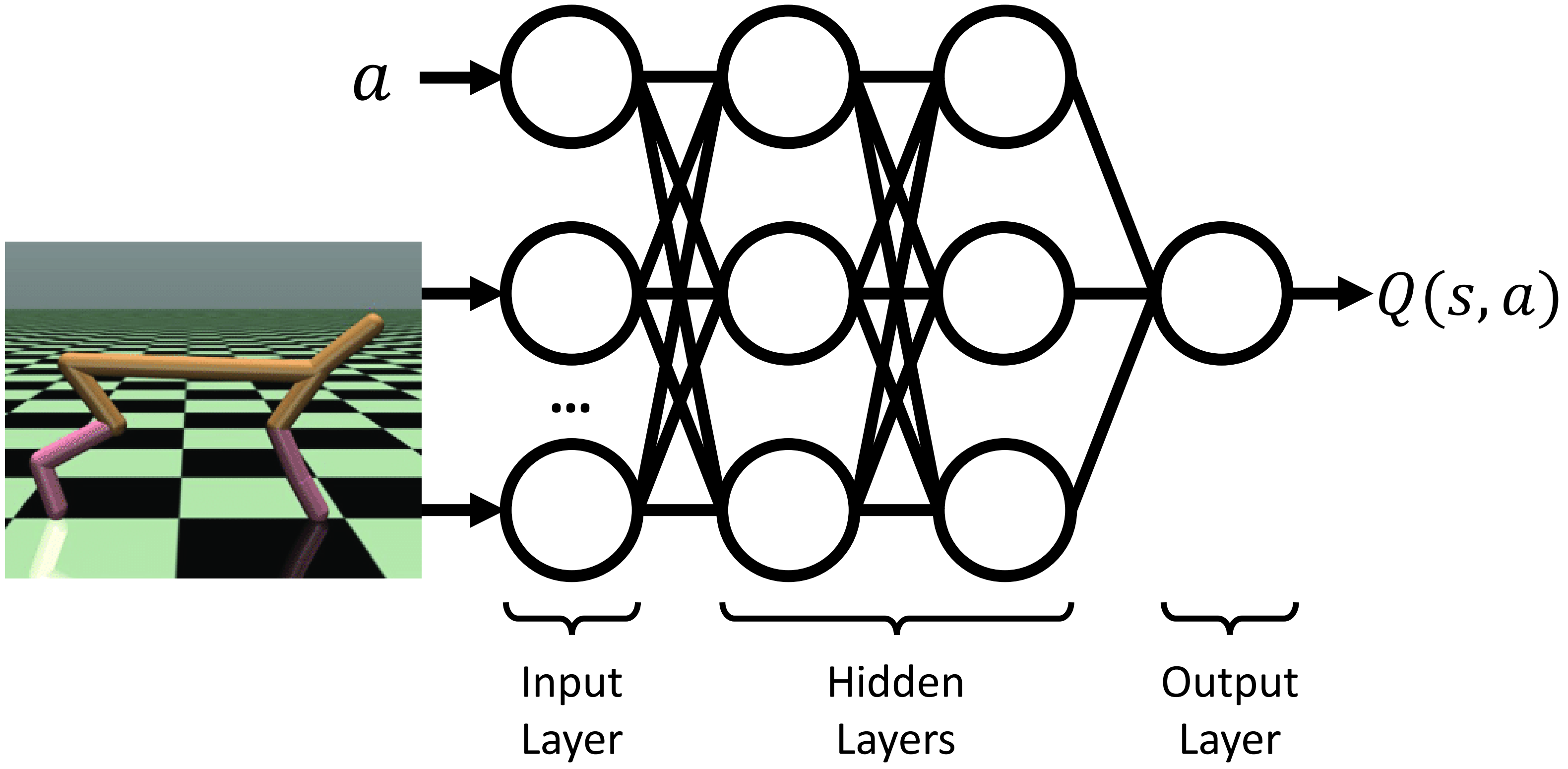
DDPG Network Architecture

Actor Network (Policy-Network)

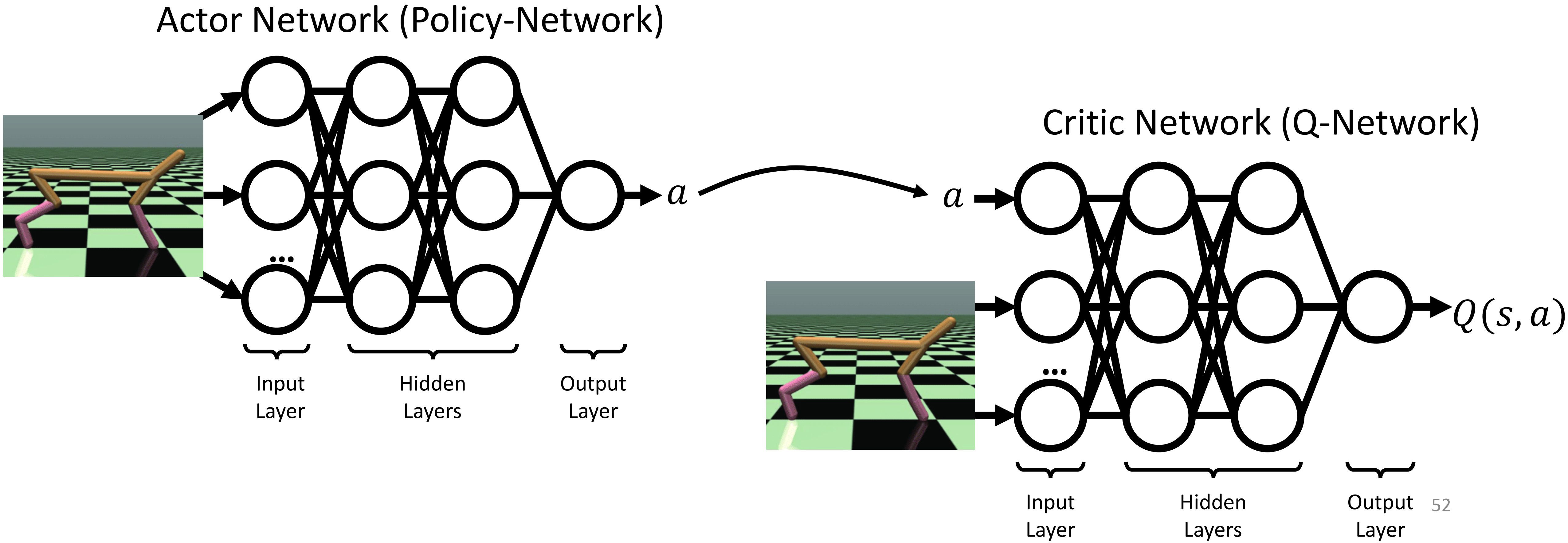


DDPG Network Architecture

Critic Network (Q-Network)



DDPG Network Architecture



Algorithm: Deep Deterministic Policy Gradients (DDPG)
Initialise replay memory D to capacity N
Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π
Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$
Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**
    Initialise random process $\mathcal{N}$ for action exploration
    Initialise initial state s_1
    For $t = 1, T$ **do**
        Select action $a_t = \pi(s_t) + \mathcal{N}_t$
        Execute action a_t and observe reward r_{t+1}, next state s_{t+1}
        Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D
        Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D
        Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$
        Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic
        Perform gradient ascent step $\nabla_{\theta^\pi}\mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor
        Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$
    End For
End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)

Initialise replay memory D to capacity N ← Pre-Fill Replay Buffer

Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π

Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$

Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**

  Initialise random process $\mathcal{N}$ for action exploration

  Initialise initial state s_1

  For $t = 1, T$ **do**

    Select action $a_t = \pi(s_t) + \mathcal{N}_t$

    Execute action a_t and observe reward r_{t+1}, next state s_{t+1}

    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D ← Store Experience in Replay Buffer

    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D ← Sample From Replay Buffer

    Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$

    Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic

    Perform gradient ascent step $\nabla_{\theta^\pi}\mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor

    Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$

  End For

End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)
Initialise replay memory D to capacity N
Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π
Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$
Initialise target network learning rate $\beta \in (0,1]$

Initialise Target Networks

For episode $= 1, M$ **do**
Initialise random process $\mathcal{N}$ for action exploration
Initialise initial state s_1
For $t = 1, T$ **do**
Select action $a_t = \pi(s_t) + \mathcal{N}_t$
Execute action a_t and observe reward r_{t+1}, next state s_{t+1}
Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D
Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D
Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$ ← Generate Target Using Target Networks
Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic
Perform gradient ascent step $\nabla_{\theta^\pi} \mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor
Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$ ← Update Target Networks
End For
End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)
Initialise replay memory D to capacity N
Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π
Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$
Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**
  Initialise random process $\mathcal{N}$ for action exploration
  Initialise initial state s_1
  For $t = 1, T$ **do**
    Select action $a_t = \pi(s_t) + \mathcal{N}_t$
    Execute action a_t and observe reward r_{t+1}, next state s_{t+1}
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D
    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D
    Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$
    Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic
    Perform gradient ascent step $\nabla_{\theta^\pi}\mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor
    Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$
  End For
End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)

Initialise replay memory D to capacity N

Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π

Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$

Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**

  $\boxed{\text{Initialise random process } \mathcal{N} \text{ for action exploration}}$

  Initialise initial state s_1

  For $t = 1, T$ **do**

    $\boxed{\text{Select action } a_t = \pi(s_t) + \mathcal{N}_t}$

    Execute action a_t and observe reward r_{t+1}, next state s_{t+1}

    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D

    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D

    Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$

    Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic

    Perform gradient ascent step $\nabla_{\theta^\pi}\mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor

    Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$

  End For

End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)
Initialise replay memory D to capacity N
Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π
Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$
Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**
  Initialise random process $\mathcal{N}$ for action exploration
  Initialise initial state s_1
  For $t = 1, T$ **do**
    Select action $a_t = \pi(s_t) + \mathcal{N}_t$
    Execute action a_t and observe reward r_{t+1}, next state s_{t+1}
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D
    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D
    Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$
    Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic
    Perform gradient ascent step $\nabla_{\theta^\pi} \mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor
    Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$
  End For
End For

Algorithm: Deep Deterministic Policy Gradients (DDPG)

Initialise replay memory D to capacity N

Initialise critic network Q with random weights θ^Q, actor network π with weights θ^π

Initialise target critic network $\hat{Q}$ with weights $\hat{\theta}^Q = \theta^Q$, target actor network $\hat{\pi}$ with weights $\hat{\theta}^\pi = \theta^\pi$

Initialise target network learning rate $\beta \in (0,1]$

For episode $= 1, M$ **do**

  Initialise random process $\mathcal{N}$ for action exploration

  Initialise initial state s_1

  For $t = 1, T$ **do**

    Select action $a_t = \pi(s_t) + \mathcal{N}_t$

    Execute action a_t and observe reward r_{t+1}, next state s_{t+1}

    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D

    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D

    Set $y_j = r_{j+1} + \gamma \hat{Q}\left(s_{j+1}, \hat{\pi}(s_{j+1})\right)$

    Perform gradient descent step $\nabla_{\theta^Q}\left(y_j - Q(s_j, a_j)\right)^2$ on critic

    Perform gradient ascent step $\nabla_{\theta^\pi}\mathrm{E}\left[Q\left(s_j, \pi(s_j)\right)\right]$ on actor

    Update target networks $\hat{\theta}^Q \leftarrow \beta\theta^Q + (1-\beta)\hat{\theta}^Q$, $\hat{\theta}^\pi \leftarrow \beta\theta^\pi + (1-\beta)\hat{\theta}^\pi$

  End For

End For

In Today's Lecture, We…

- Introduced policy-based methods as an alternative to value-based methods.
- Discussed why policy-based method are useful for solving problems where the optimal policy is stochastic.
- Derived the policy-gradient from first principles.
- Looked at a concrete Monte-Carlo Policy-Gradient algorithm, REINFORCE.
- Introduced **baseline functions** as a method of reducing the variance in our policy gradient updates.
- Introduced **actor-critic methods**, which combine ideas from both value-based and policy-based RL methods.
- Looked at a concrete actor-critic algorithm, **DDPG**.

Acknowledgements

- Chapter 13, Reinforcement Learning (2nd Ed.), Sutton & Barto 2018
- David Silver's Policy Gradient Lecture
- OpenAI Spinning Up
  - OpenAI Spinning Up Policy Gradient Derivation
- REINFORCE Paper, Williams 1992(!)
- DDPG Paper, Lillicrap et al. 2016