Reinforcement Learning: Policy Gradient Algorithms

Earl Wong

The Policy

- In a fully observable Markov Decision Process (MDP), the optimal value function can be used to determine the optimal deterministic policy.

- In practice, MDP's are not guaranteed to be fully observable.

- i.e. Partially Observable Markov Decision Processes (POMDP) also exist.

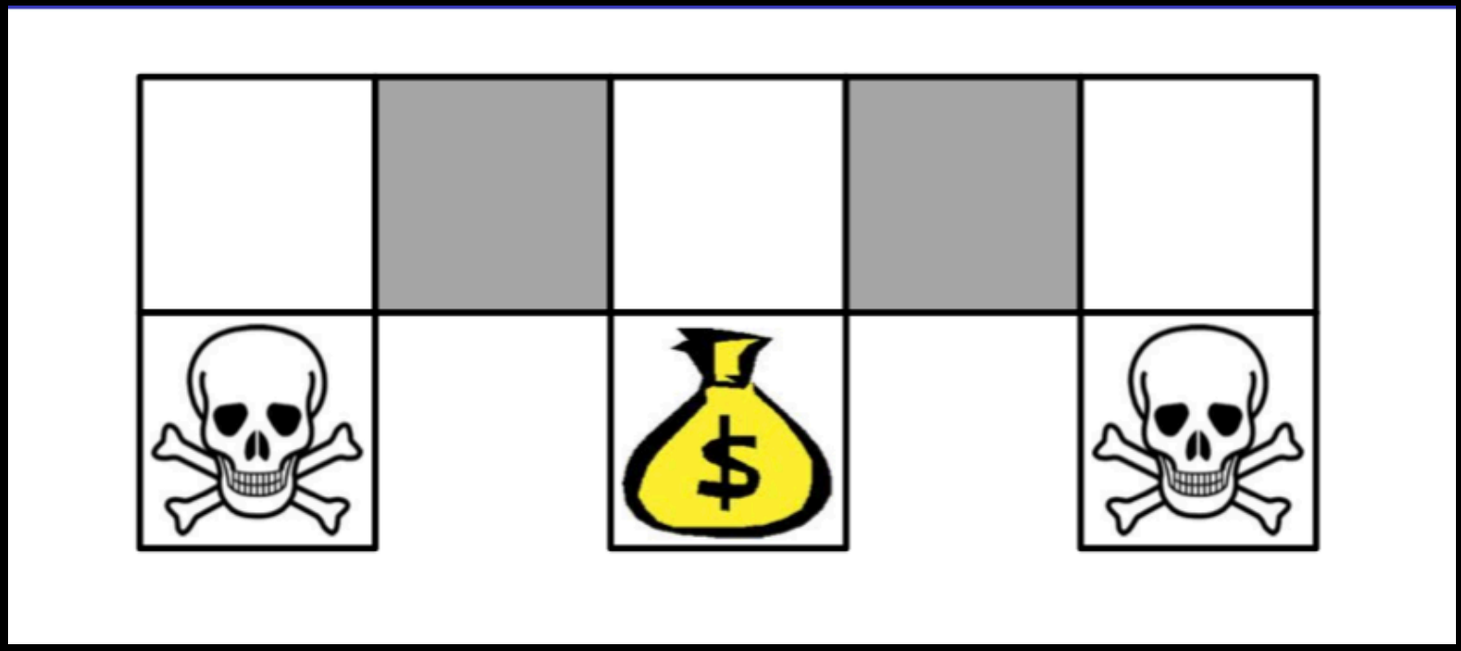
- In practice, partially Observable Markov Decision Processes can result from a variety of causes: aliased states, the use of function approximators, etc.

- What is the best policy for a POMDP?

The Need for a Stochastic Policy

- In POMDP's, stochastic policies can be useful.
- Consider the game of "rock, paper, scissors."
  - If a deterministic policy is used (i.e. pick rock every time), the deterministic policy will be easily exploited by an opponent who always chooses paper.
  - In contrast, if a uniform random policy is used, the expected return will be maximized.

The Need for a Stochastic Policy



Above, is a toy problem illustrating the concept of aliased states.

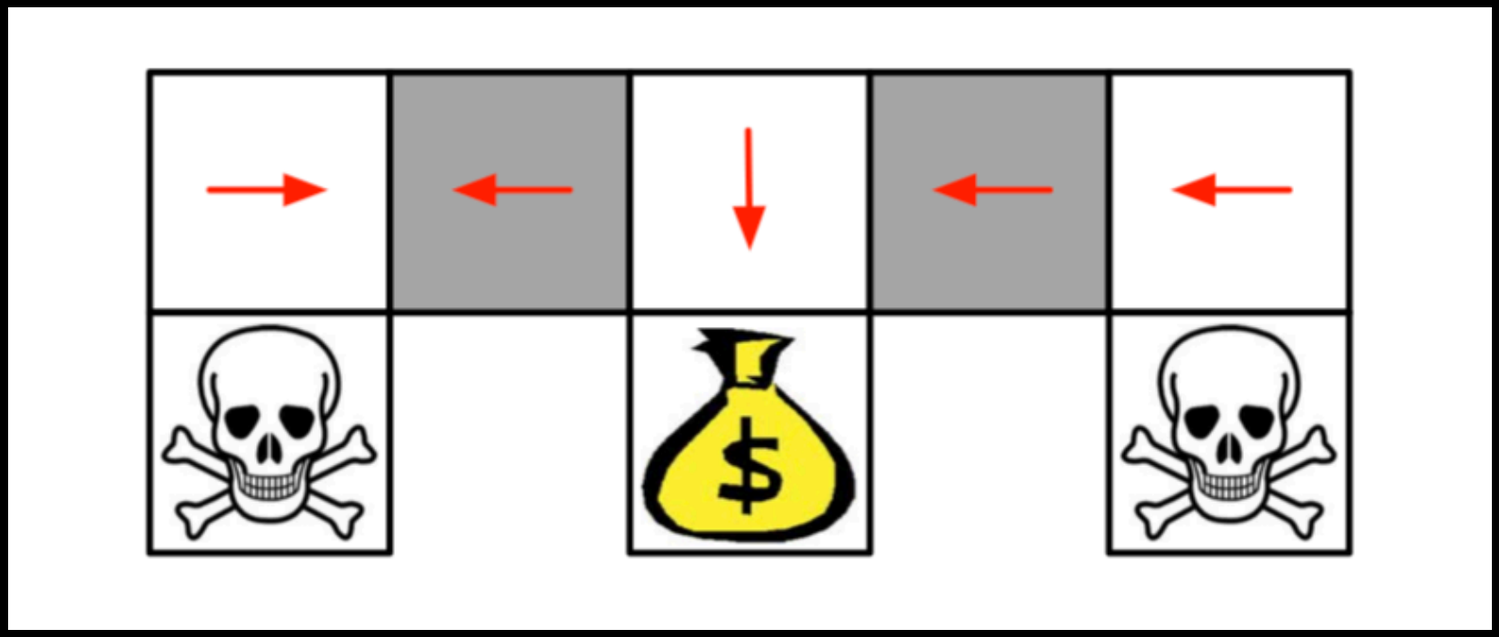
Suppose the permissible actions for the gray states are: move N, S, E, W

By restricting our actions to these four directions, the gray squares become “identical”.

i.e. They are aliased.

Figure taken from S. Levine lecture notes.

The Need for a Stochastic Policy

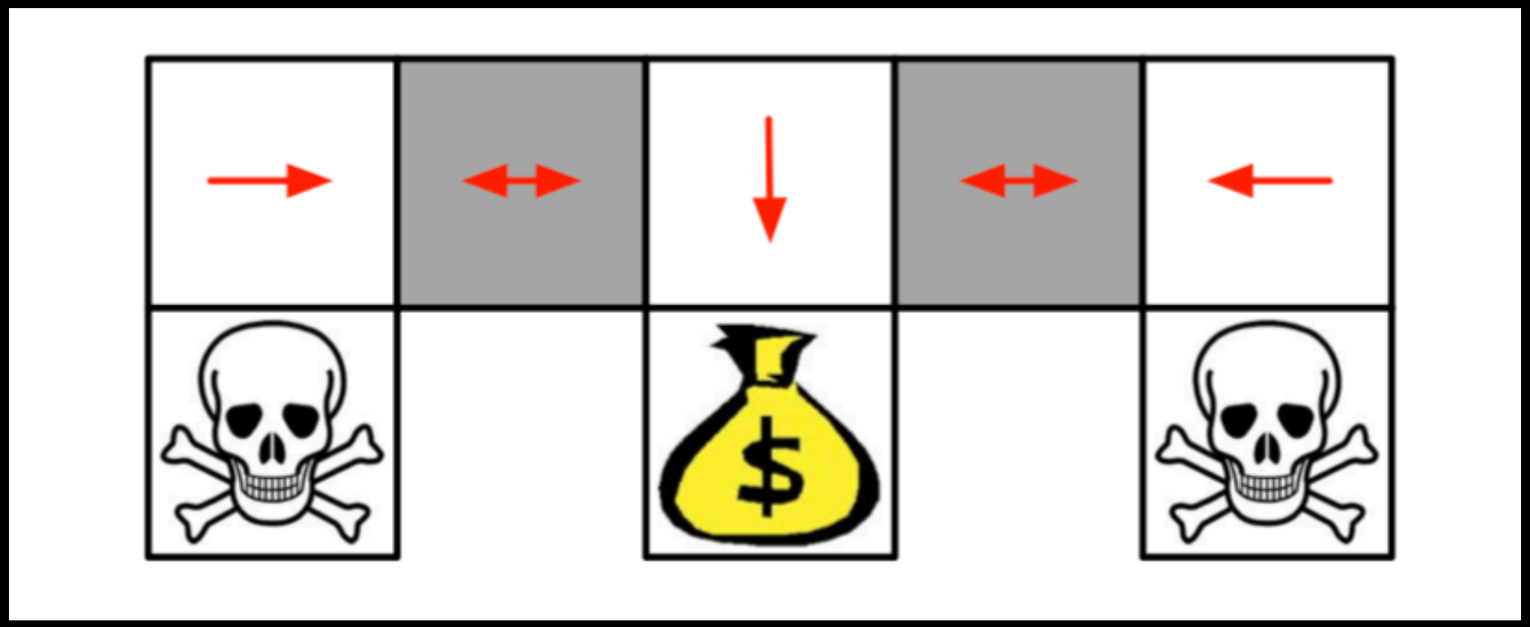


In this diagram, the red arrows indicate the actions associated with a deterministic policy.

If an agent starts (or ends up) in the left gray square, the agent will never reach the terminal state containing the money.

If an agent starts in the left gray square and an epsilon greedy policy is used, the agent eventually reaches the terminal state containing the money after many, many attempts.

The Need for a Stochastic Policy



In this diagram, the red arrows in the gray squares indicate a 50 / 50 stochastic policy.

Now, if an agent lands in either of the gray squares, the agent will reach the terminal state containing the money in an efficient manner, since it will never “get stuck”.

This toy problem illustrates the benefits of using a stochastic policy.

Policy Gradients

- Suppose a neural network is used to create a policy.
- How would the neural network learn the policy?
- i.e. How should the weights of the neural network be modified, in order to produce the maximum expected return?
- To answer this question, we need to define an objective function.

Policy Gradients

- Let $J(\theta)$ be an objective function that depends on a set of weight parameters θ
- In order to maximize the objective function, an algorithm will need to move the weight parameters θ to a set of "optimal values".
- This can be accomplished by computing partial derivatives of the objective function with respect to each weight parameter: $$\nabla_\theta J(\theta) = (\frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \ldots, \frac{\partial J(\theta)}{\partial \theta_n})$$
- and updating the weight parameters using the following update rule: $$\theta_{new} = \theta_{old} + \alpha \nabla_\theta J(\theta)$$

Example (Small Perturbations): Empirically Training AIBO To Walk

- AIBO is a mechanical walking dog, whose walk policy is controlled by 12 parameters.
- To find the optimal parameter settings, each parameter is individually tuned.
- First, parameter one is increased by some “delta”, and the walking speed is observed.
- If the walking speed becomes slower, parameter one is repeatedly decreased by some “delta”, until the walking speed increases.
- Next, parameter two is increased by some “delta”, and the walking speed is observed ….
- This process repeats (again and again) for all 12 parameters, until the walking speed is maximized.

Policy Gradients

- An analytic approach can also be used.
- Let the desired policy be given by a function approximator $\pi_\theta(a|s)$ where $\pi_\theta(a|s) = Pr[a|s, \theta]$
- For the analytic solution, $\nabla_\theta \pi_\theta(a|s)$ needs to be computed.
- This is given by $\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s) \nabla_\theta log(\pi_\theta(a|s))$
- The second term of the equation - $\nabla_\theta log(\pi_\theta(a|s))$ - is called the score function.

Policy Gradient

Now, let's evaluate the policy gradient.

First let's define a trajectory / MC rollout:

$$p_\theta(\tau) = p_\theta(s_1, a_1, \ldots, \ldots s_T, a_T)$$

$$p_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t,\ a_t)$$

Next, let's define an objective function and it's argmax:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t})$$

$$\theta^* = argmax_\theta J(\theta)$$

$$\theta^* = argmax_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

$$\theta^* = argmax_\theta E_{\tau \sim p_\theta(\tau)} [r(\tau)]$$

Policy Gradient

Returning to the objective function:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t})$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau$$

We can now compute the gradient:

$$\nabla_\theta J(\theta) = \int \nabla_\theta (p_\theta(\tau)) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta (log(p_\theta(\tau)) r(\tau) d\tau$$

Recall:

$$log(p_\theta(\tau)) = log(p(s_1)) + \sum_{t=1}^{T} log(\pi_\theta(a_t|s_t)) + \sum_{t=1}^{T} log(p(s_{t+1}|s_t, a_t))$$

Where:

$$\nabla_\theta (log(p_\theta(\tau))) = \nabla_\theta \left[log(p(s_1)) + \sum_{t=1}^{T} log(\pi_\theta(a_t|s_t)) + \sum_{t=1}^{T} log(p(s_{t+1}|s_t, a_t)) \right]$$

Policy Gradient

Substituting:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \sum_{t=1}^{T} \nabla_\theta log(\pi_\theta(a_t|s_t)) r(\tau) d\tau$$

Which simplifies to:

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta (log(\pi_\theta(a_t|s_t))) \sum_{t=1}^{T} r(s_t, a_t) \right]$$

Approximating the expectation by averaging the sample rollouts, we obtain:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta (log(\pi_\theta(a_{i,t}|s_{i,t}))) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \right]$$

Finally, we perform our update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

REINFORCE
(The Simplest Policy Gradient Algorithm)

Initialize the weights of a neural network.

For each episodic rollout (N=1) or a batch of episodic rollouts (N<>1):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta(log\acute{\pi}_\theta(a_{i,t}|s_{i,t}))\right)\left(\sum_{t=1}^{T}r(s_{i,t},a_{i,t})\right)\right]$$

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$

Instead of calculating the return from the beginning of time, Rewards to Go says that "only the rewards occurring after time t for the current action a_t **are relevant".**

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta(log\acute{\pi}_\theta(a_{i,t}|s_{i,t}))\right)\left(\sum_{t'=t}^{T}r_{s_{i,t'},a_{i,t'}}\right)\right]$$

Using the Rewards to Go return, a better policy results, due to the reduced variance in the computed returns.

Reducing the Variance

- In the current realization using raw MC rollouts, the outputs have high variance.
- High variance outputs are undesirable, because a large number of rollouts are required to decrease the variance and / or find a good policy.
- How can the output variance be reduced?
- “Rewards To Go” was one way.
- Another way to reduce the output variance, is to subtract a constant baseline from each output state.

Reducing the Variance

- For example, let the outputs be 10, 20, 50, 30 and 40.
- The corresponding mean (for the above values) is 30.
- Now, subtract a baseline (=30) from every output value and THEN apply the new value: {(10-30), (20-30), (50-30), (30-30) and (40-30)}
- In general, any baseline can be subtracted (without unwanted effects), as long as the baseline is not a function of θ
- This is because $\nabla_\theta B(s) = 0$, when the baseline is not a function of θ

Employing a Critic

- Hence, an even better way to reduce the output variance is to estimate a state value function V(s), and to let B(s) = V(s).

- The state value function V(s) needs to be learned from the policy.

- When policy estimation and state value function estimation occur “under one roof”, we obtain the actor-critic algorithm architecture.

Employing a Critic

- Employing the baseline subtraction trick and substituting the action value function for return, we obtain:

$$A_{\pi_\theta}(s,\, a) = Q_{\pi_\theta}(s,\, a) - V_{\pi_\theta}(s)$$

- The advantage function produces a very low variance output.

- However, we now (also) need to estimate a new set of parameters associated with the state value function.

Actor Critic Algorithm

- Actor critic algorithms require several sets of parameters to be computed and stored.
- One set of parameters, θ , is associated with the policy (= actor).
- If we estimate the action value function $Q_{\pi_\theta}(s,a) \sim Q_w(s,a)$, we would also need to estimate a set of parameters w.
- For the state value function $V_{\pi_\theta}(s) \sim V_v(s)$ (=critic), we would need to estimate a set of parameters v.
- Note: Action value function $Q_{\pi_\theta}(s,a) \sim Q_w(s,a)$ (=critic) is also a perfectly acceptable substitution.

Actor Critic Algorithm For Action Value Function

$$Initialize\ s, \theta\ and\ w$$

$$a \sim \pi_\theta(a \mid s)$$

do:

Obtain r(s,a) and s'

$$a' \sim \pi_\theta(a \mid s')$$

$$\delta_{TD} = r(s,a) + \gamma Q_w(s',a') - Q_w(s,a)$$ } **TD(0) Advantage**

$$\theta \leftarrow \theta + \alpha \cdot \delta_{TD} \nabla_\theta log(\pi_\theta(a \mid s))$$ } **Actor update**

$$w \leftarrow w + \beta \cdot \delta_{TD}^2$$ } **Critic update**

$$s \leftarrow s'\ and\ a \leftarrow a'$$

Different Forms of Policy Gradient

REINFORCE

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t \right]$$

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

Actor Critic

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A_t \right]$$

$$A_t = G_t - V_v(s_t)$$

TD Actor Critic

$$A_t = r_t + \gamma V_v(s_{t+1}) - V_v(s_t)$$

The following slide was taken from David Silver's lecture notes, and summarizes the presented knowledge, up to this point.

- 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