

“The test of a man isn’t what you think he’ll do. It’s what he actually does.”

Frank Herbert, *Dune*

A desert landscape with sand dunes under a bright sun, with silhouettes of people on a dune.

CMPUT 365

Introduction to RL

Marlos C. Machado

<https://openart.ai/discovery/sd-1006656316309258270>

Classes 33-35/36

Coursera Reminder

You **should be enrolled in the private session** we created in Coursera for CMPUT 365.

I **cannot** use marks from the public repository for your course marks.

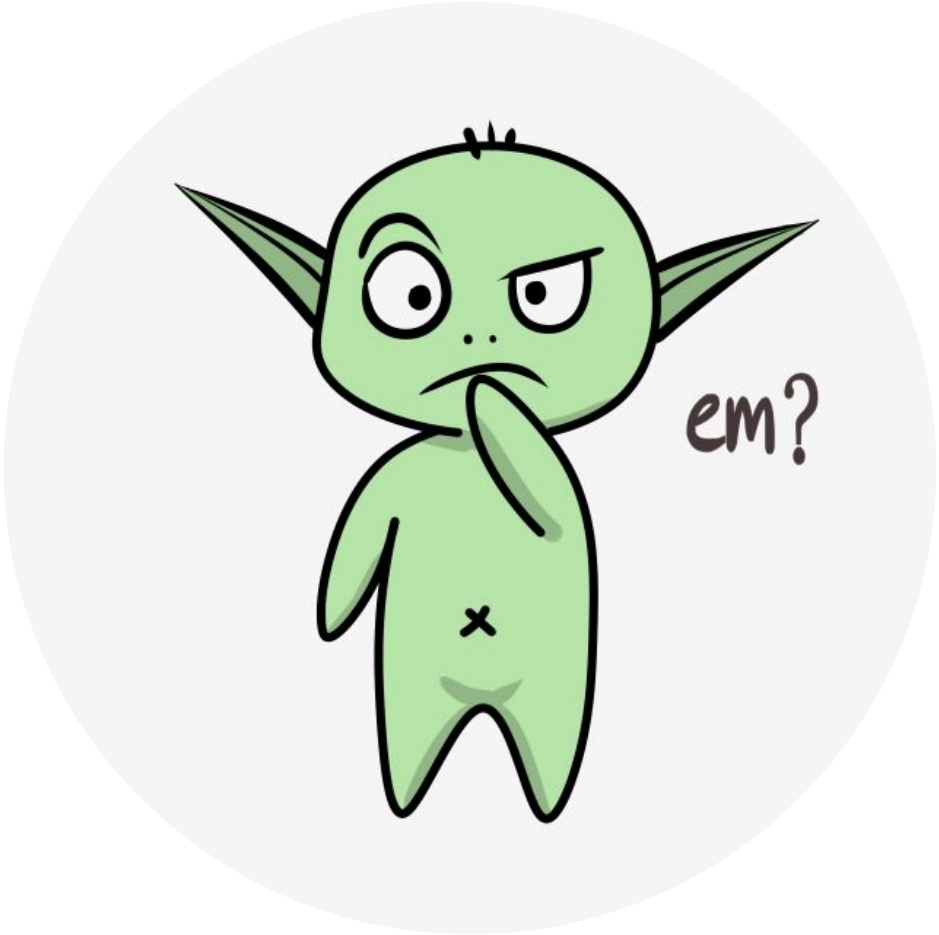
You **need** to **check, every time**, if you are in the private session and if you are submitting quizzes and assignments to the private section.

The deadlines in the public session **do not align** with the deadlines in Coursera.

If you have any questions or concerns, **talk with the TAs** or email us
`cmput365@ualberta.ca`.

Reminders and Notes

- Your last quiz and programming assignments are due on Friday.
- A note on the final exam:
 - The required reading from the syllabus does not mean that's what will be covered in the final exam. There are some mismatches. Anything we discussed in class is fair game, including Maximization Bias and Double Learning (Sec. 6.7), and Nonlinear Function Approximation: Artificial Neural Networks (Sec. 9.7).
 - Final will be *2 hours long*, and questions will cover the whole term.
- SPOT Survey is still available for you.



Chapter 13

Policy Gradient Methods

Policy Gradient Methods

- Pretty much everything so far has been about action-value methods.
 - They learn value functions and then select an action based on their estimated action values.
- What if we learned the policy directly?

Policy Gradient Methods– Why?

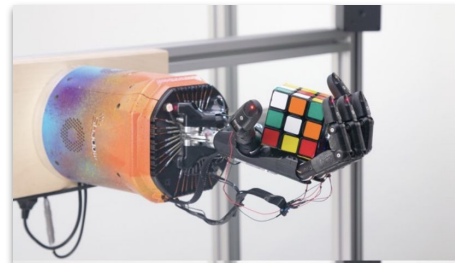
- Everything we discussed so far assumed we had a discrete action set.
 - Many problems we care about have an action set with continuous actions (e.g., torque in a motor).
- It naturally “scales” to function approximation and sometimes the PG algorithms have much nicer guarantees.
- Maybe one should directly optimize what they care about, which is the policy.
- It works and it is used everywhere now $\backslash_(\text{ツ})_/_$



LLMs (e.g., ChatGPT)



Video games
(e.g., Dota2, StarCraft 2)



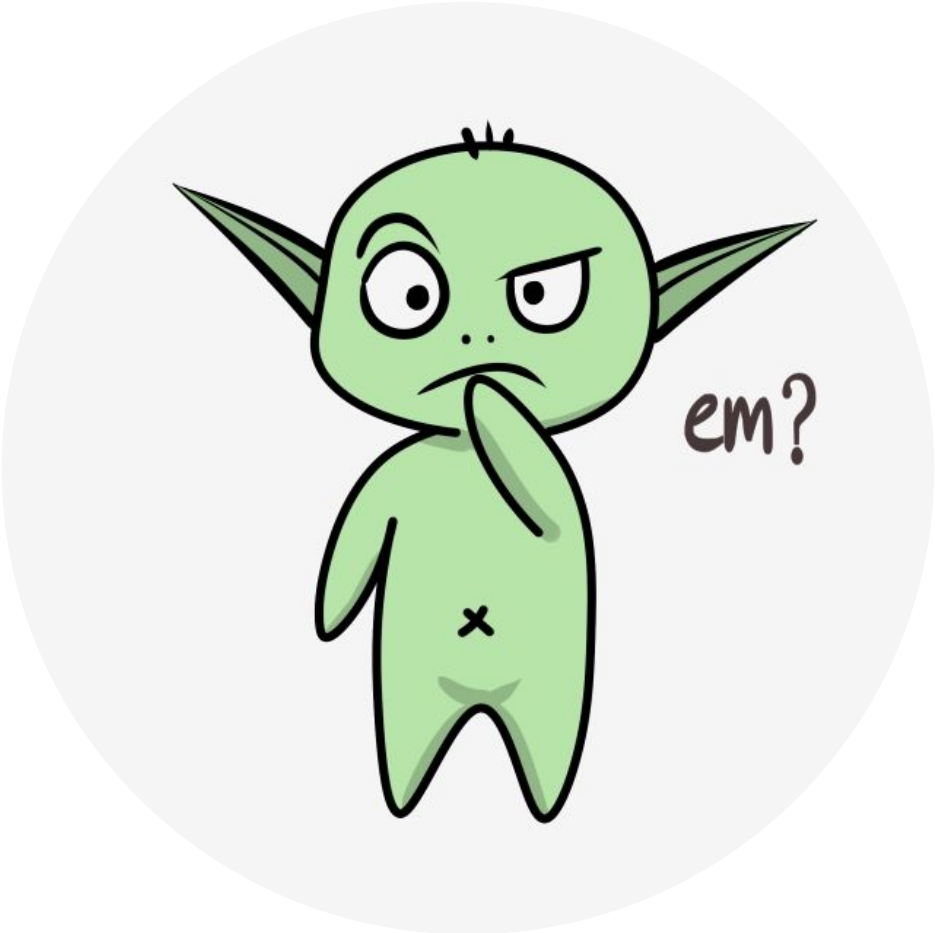
Robotics!



More Specifically

- Let $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ denote the policy's parameter vector. We then write:
 $\pi(a | s, \boldsymbol{\theta}) = \Pr(A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$.
- We consider some scalar performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameter. We perform gradient *ascent* in J :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$



Gradient Bandit Algorithms (Chapter 2) – No parameterization

- Bandits! But instead of learning an estimate of the expected reward from each arm, we learn a numerical *preference* for each action a , denoted $H_t(a)$.
 - The larger the preference the more likely you are to take that action, but the preference has no additional semantics in terms of rewards.
 - If we offset all action preferences we will still select actions with the same probability.
- We choose actions according to a *softmax distribution* (i.e., Gibbs or Boltzmann distribution):

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \doteq \pi_t(a)$$

At first, all action preferences are the same

Notice we are not conditioning on s , because it is a bandits problem

Some more on the Softmax

$$\frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}}$$

| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | 1 | 1 | 1 |
| π(·) | 0.33 | 0.33 | 0.33 |

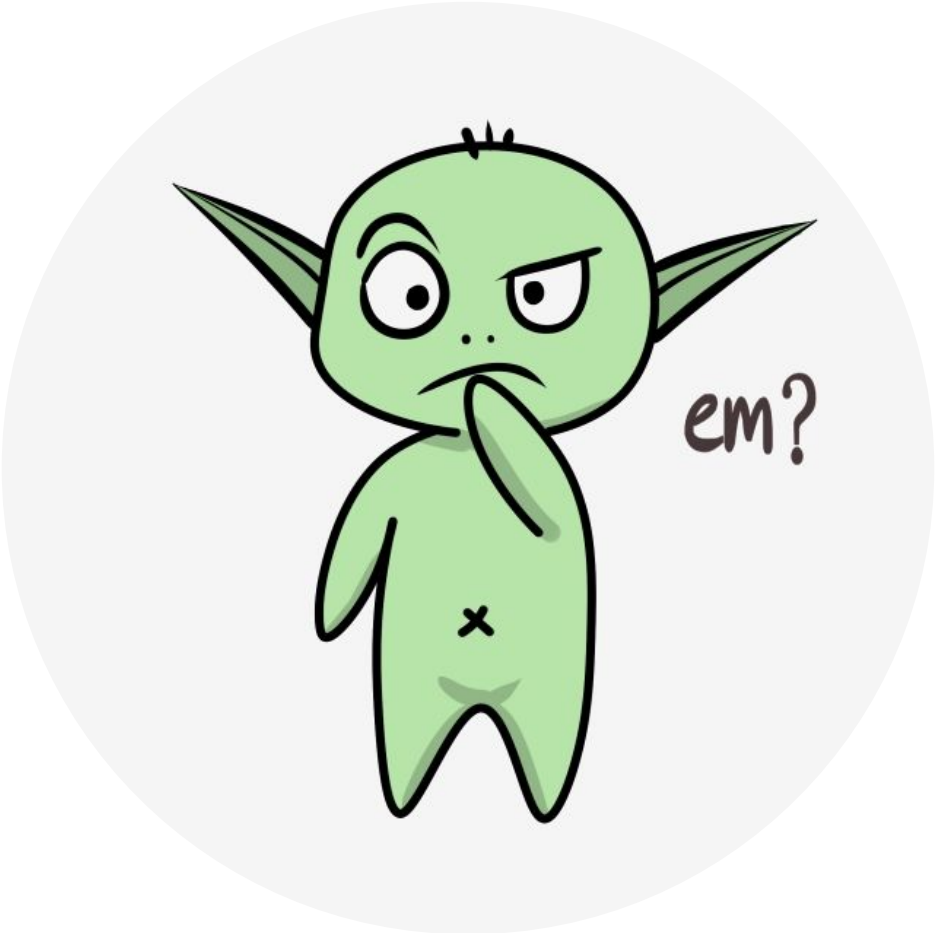
| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | 4 | 1 | 1 |
| π(·) | 0.91 | 0.05 | 0.05 |

| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | 3 | 2 | 1 |
| π(·) | 0.67 | 0.24 | 0.09 |

| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | 3 | 1 | 1 |
| π(·) | 0.79 | 0.11 | 0.11 |

| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | -4 | 1 | 1 |
| π(·) | 0 | 0.5 | 0.5 |

| | a₁ | a₂ | a₃ |
|-------------|----------------------|----------------------|----------------------|
| H(·) | 4 | 3 | 2 |
| π(·) | 0.67 | 0.24 | 0.09 |



Gradient Bandit Algorithms (Chapter 2) – No parameterization

- We change the preferences with stochastic gradient ascent.
- The idea:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$$

where the measure of performance here is the expected reward:

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

Obviously, we don't know q_* .

... but we can be clever about it.

Gradient Bandit Algorithms (Chapter 2) – Derivation

Let's look at the exact performance gradient:

$$\begin{aligned}\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] \\ &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)}\end{aligned}$$

Next, we multiply each term of the sum by $\pi_t(x)/\pi_t(x)$:

$$\begin{aligned}\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \sum_x \pi_t(x) q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \bigg/ \pi_t(x) \\ &= \mathbb{E} \left[q_*(A_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \bigg/ \pi_t(A_t) \right] = \mathbb{E} \left[R_t \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \bigg/ \pi_t(A_t) \right]\end{aligned}$$

This is an expectation, we are summing over all possible values x of the random variable A_t , then multiplying by the probability of taking those values.

Because, $\mathbb{E}[R_t | A_t] = q_*(A_t)$.

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E} \left[R_t \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \middle/ \pi_t(A_t) \right]$$

Gradient Bandit Algorithms (Chapter 2) – Derivation

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \pi(x)$$

Let's instantiate $\pi_t(x)$: $\pi_t(x) = \frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}}$

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}}$$

$$= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^k e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)} \right)^2}$$

Quotient rule:

$$\frac{\partial}{\partial x} \left[\frac{f(x)}{g(x)} \right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x) \frac{\partial g(x)}{\partial x}}{g(x)^2}$$

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E} \left[R_t \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \middle/ \pi_t(A_t) \right]$$

Gradient Bandit Algorithms (Chapter 2) – Derivation

$$\begin{aligned} \frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^k e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)} \right)^2} \\ &= \frac{\mathbb{1}_{a=x} e^{H_t(x)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)} \right)^2} \end{aligned}$$

Recall:

$$\frac{\partial e^x}{\partial x} = e^x$$

Thus:

$$\frac{\partial e^{H_t(x)}}{\partial H_t(a)} = \mathbb{1}_{a=x} e^{H_t(x)}$$

that is, when $x = y$, we have the identity, o.w. we have zero.

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E} \left[R_t \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \middle/ \pi_t(A_t) \right]$$

Gradient Bandit Algorithms (Chapter 2) – Derivation

$$\begin{aligned} \frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\mathbb{1}_{a=x} e^{H_t(x)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)} \right)^2} \\ &= \frac{\mathbb{1}_{a=x} e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} - \frac{e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)} \right)^2} \\ &= \mathbb{1}_{a=x} \pi_t(x) - \pi_t(x) \pi_t(a) \\ &= \pi_t(x) (\mathbb{1}_{a=x} - \pi_t(a)) \end{aligned}$$

Gradient Bandit Algorithms (Chapter 2) – Derivation

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E} \left[R_t \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \middle/ \pi_t(A_t) \right]$$

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbf{1}_{a=x} - \pi_t(a))$$

$$= \mathbb{E} \left[\frac{R_t \pi_t(A_t) (\mathbf{1}_{a=A_t} - \pi_t(a))}{\pi_t(A_t)} \right]$$

$$= \mathbb{E} \left[R_t (\mathbf{1}_{a=A_t} - \pi_t(a)) \right]$$

Gradient Bandit Algorithms (Chapter 2) – Baseline

Let's look at the exact performance gradient:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] = \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)}$$

Instead, we could do:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} = \sum_x \left(q_*(x) - \bar{R}_t \right) \frac{\partial \pi_t(x)}{\partial H_t(a)}$$

Baseline: Any scalar that does not depend on x .

We can do this because the gradient sums to zero over all the actions, $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$. As $H_t(a)$ is changed, some actions' prob. go up and some go down, but the sum of the changes must be zero because the sum of the prob. is always 1.

Next, we multiply each term of the sum by $\pi_t(x)/\pi_t(x)$:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x) \left(q_*(x) - \bar{R}_t \right) \frac{\partial \pi_t(x)}{\partial H_t(a)} \bigg/ \pi_t(x) = \mathbb{E} \left[\left(q_*(x) - \bar{R}_t \right) \frac{\partial \pi_t(x)}{\partial H_t(a)} \bigg/ \pi_t(x) \right] = \mathbb{E} \left[\left(R_t - \bar{R}_t \right) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \bigg/ \pi_t(A_t) \right]$$

Average of the rewards up to but not including time t .

Gradient Bandit Algorithms (Chapter 2) – No parameterization

- We change the preferences with stochastic gradient ascent.
- The idea:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} \quad \text{where} \quad \mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

with our derivation, we then have:

$$H_{t+1}(a) = H_t(a) + \alpha \mathbb{E} \left[R_t \left(\mathbf{1}_{a=A_t} - \pi_t(a) \right) \right]$$

Which means:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha R_t (1 - \pi_t(A_t)) \quad \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha R_t \pi_t(a) \quad \text{for all } a \neq A_t. \end{aligned}$$



Policy Gradient Methods

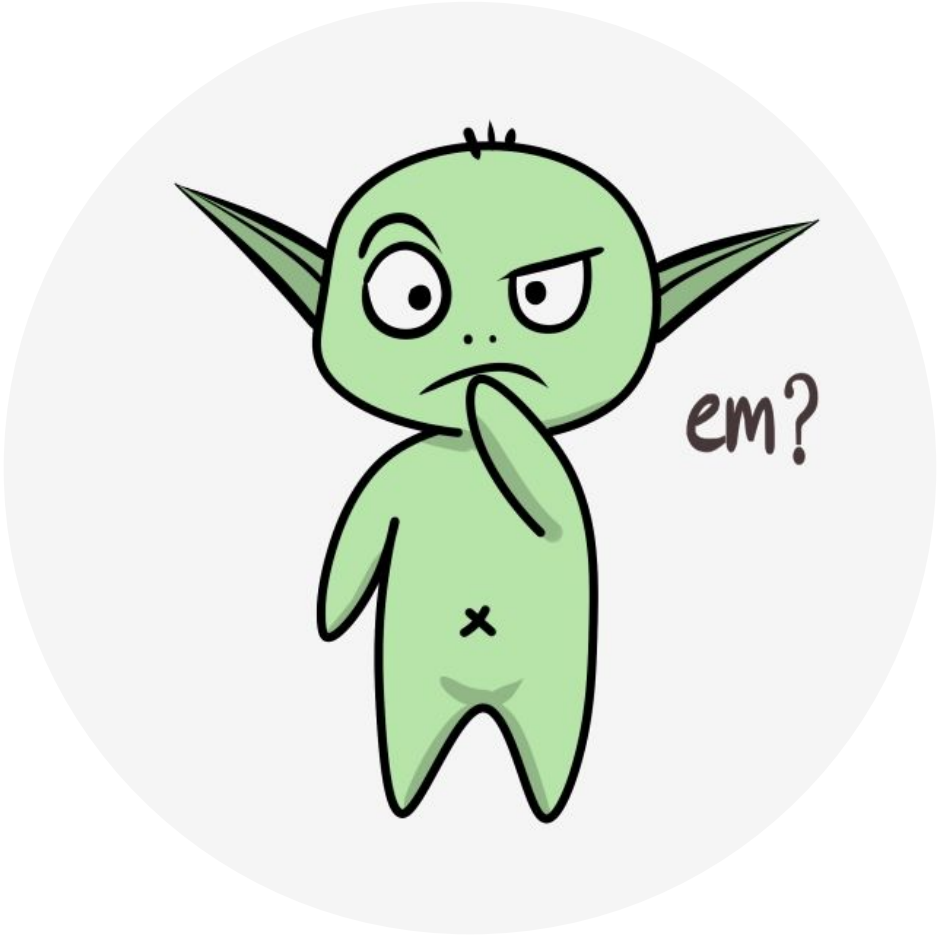
- Pretty much everything so far has been about action-value methods.
 - They learn value functions and then select an action based on their estimated action values.
- What if we learned the policy directly?
 - We parameterize a policy and we learn the values of those parameters π_θ
 - We can still use a value function to learn the policy params, but it isn't required for action selection.

More Specifically

- Let $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ denote the policy's parameter vector. We then write:
 $\pi(a | s, \boldsymbol{\theta}) = \Pr(A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$.
- We consider some scalar performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameter. We perform gradient *ascent* in J :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

- Methods that learn approximations to both policy and value functions are often called actor-critic methods.
 - Actor: learned policy
 - Critic: learned value function



Scaling up and Generalizing Policy Gradient Methods

- In the Bandits problem, we used the following softmax:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \doteq \pi_t(a)$$

- More generally, we need to introduce states and to parameterize H_t :

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

This could be “anything”, such as a neural network or linear in features:

$$e^{h(s,a,\boldsymbol{\theta})} = e^{\boldsymbol{\theta}^\top \mathbf{x}(s,a)}$$

Some Advantages of Policy Approximation

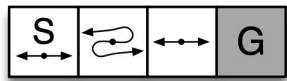
- The softmax in action preferences can approach a deterministic policy, whereas ϵ -greedy, for example, will always have a probability ϵ of acting suboptimally.
 - These decisions are made per “state” with policy gradient methods, not globally.

Some Advantages of Policy Approximation

- The softmax in action preferences can approach a deterministic policy, whereas ϵ -greedy, for example, will always have a probability ϵ of acting suboptimally.
 - These decisions are made per “state” with policy gradient methods, not globally.
- It is not the softmax itself that gives us that, but the use of action preferences. A softmax over state-action values would never converge to a deterministic policy.
 - Action preferences are driven to produce the optimal *stochastic* policy.

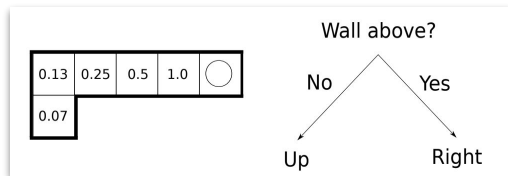
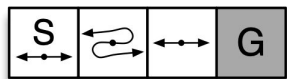
Some Advantages of Policy Approximation

- The softmax in action preferences can approach a deterministic policy, whereas ϵ -greedy, for example, will always have a probability ϵ of acting suboptimally.
 - These decisions are made per “state” with policy gradient methods, not globally.
- It is not the softmax itself that gives us that, but the use of action preferences. A softmax over state-action values would never converge to a deterministic policy.
 - Action preferences are driven to produce the optimal *stochastic* policy.
- We can select actions with arbitrary probabilities. Sometimes the best policy is a stochastic one.
 - See Example 13.1 of the textbook

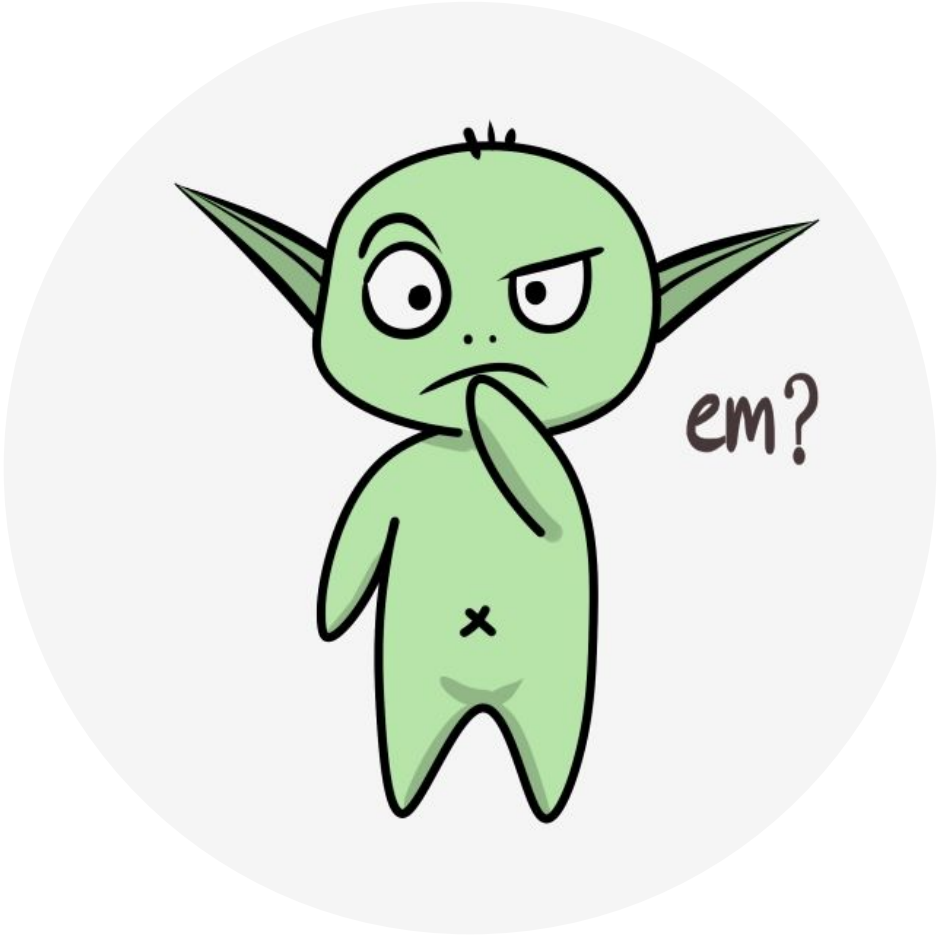


Some Advantages of Policy Approximation

- The softmax in action preferences can approach a deterministic policy, whereas ϵ -greedy, for example, will always have a probability ϵ of acting suboptimally.
 - These decisions are made per “state” with policy gradient methods, not globally.
- It is not the softmax itself that gives us that, but the use of action preferences. A softmax over state-action values would never converge to a deterministic policy.^z
 - Action preferences are driven to produce the optimal *stochastic* policy.
- We can select actions with arbitrary probabilities. Sometimes the best policy is a stochastic one.
 - See Example 13.1 of the textbook
- A policy might be a easier to approximate than the VF. And it is the thing you ultimately care about _(ツ)_/



[Das Gupta, Talvitie, Bowling; AAAI 2015]



The Policy Gradient Theorem

- We have stronger convergence guarantees for policy gradient methods, in part because the policy changes more smoothly than, say, ε -greedy policies.
- Let's do gradient ascent, in the full RL problem. To do that we need to define $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

- It seems tricky though. “The problem is that performance depends on both the action selections and the distribution of states in which those selections are made, and that both of these are affected by the policy parameter.”
 - The effect of a change in the policy on the state distribution is typically unknown.

The Policy Gradient Theorem

- We have stronger convergence guarantees for policy gradient methods, in part because the policy changes more smoothly than, say, ε -greedy policies.
- Let's do gradient ascent, in the full RL problem. To do that we need to define $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

- It seems tricky though. “The problem is that performance depends on both the action selections and the distribution of states in which those selections are made, and that both of these are affected by the policy parameter.”
 - The effect of a change in the policy on the state distribution is typically unknown.

How can we estimate the performance gradient w.r.t the policy parameter when the gradient depends on the unknown effect of policy changes on the state distribution?

The Policy Gradient Theorem

- The Policy Gradient Theorem [Marbach and Tsitsiklis, 1998, 2001; Sutton et al. 2000] provides an analytic expression for the gradient of performance w.r.t. the policy parameter that does not involve the derivative of the state distribution.

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta})$$

Proportional to

On-policy distribution

Interlude: The On-Policy Distribution (Page 199)

[In episodic tasks, without discounting.]

- In an episodic task, the on-policy distribution depends on the initial state distribut.
- Let $h(s)$ denote the probability that an episode begins in each state s .
- Let $\eta(s)$ denote the # of time steps spent, on avg, in state s in a single episode.
Time is spent in a state s if episodes start in s , or if transitions are made into s from a preceding state \bar{s} in which time is spent.

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a|\bar{s}) p(s|\bar{s}, a), \quad \text{for all } s \in \mathcal{S}.$$

Interlude: The On-Policy Distribution (Page 199)

[In episodic tasks, without discounting.]

- In an episodic task, the on-policy distribution depends on the initial state distribut.
- Let $h(s)$ denote the probability that an episode begins in each state s .
- Let $\eta(s)$ denote the # of time steps spent, on avg, in state s in a single episode.
Time is spent in a state s if episodes start in s , or if transitions are made into s from a preceding state \bar{s} in which time is spent.

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a|\bar{s}) p(s|\bar{s}, a), \quad \text{for all } s \in \mathcal{S}.$$

- Solving this system of equations we obtain the on-policy distribution, which is the fraction of time spent in each state normalized to sum to one:

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \quad \text{for all } s \in \mathcal{S}.$$

The Policy Gradient Theorem – Proof

[For simplicity, we leave it implicit that π is a function of θ , and all gradients are also implicitly with respect to θ .]

$$\nabla v_{\pi}(s) = \nabla \left[\sum_a \pi(a|s) q_{\pi}(s, a) \right], \quad \text{for all } s \in \mathcal{S}$$

$$= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \nabla q_{\pi}(s, a) \right]$$

$$= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \nabla \sum_{s', r} p(s', r|s, a) (r + v_{\pi}(s')) \right]$$

$$= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \nabla v_{\pi}(s') \right]$$

$$= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \sum_{a'} \left[\nabla \pi(a'|s') q_{\pi}(s', a') + \pi(a'|s') \sum_{s''} p(s''|s', a') \nabla v_{\pi}(s'') \right] \right]$$

Reminder

Product rule

$$\nabla(fg) = f \nabla g + g \nabla f$$

The Policy Gradient Theorem – Proof

[For simplicity, we leave it implicit that π is a function of θ , and all gradients are also implicitly with respect to θ .]

$$\nabla v_{\pi}(s) = \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \sum_{a'} \left[\nabla \pi(a'|s') q_{\pi}(s', a') + \pi(a'|s') \sum_{s''} p(s''|s', a') \nabla v_{\pi}(s'') \right] \right]$$

$k=0$ (above the first term) $k=1$ (above the second term)

If we keep unrolling, considering all time steps:

$$= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \rightarrow x, k, \pi) \sum_a \nabla \pi(a|x) q_{\pi}(x, a)$$

Probability of transitioning from state s to state x in k steps under policy π .

The Policy Gradient Theorem – Proof

[For simplicity, we leave it implicit that π is a function of θ , and all gradients are also implicitly with respect to θ .]

$$\begin{aligned}
 \nabla J(\theta) &= \nabla v_\pi(s) \\
 &= \sum_s \left(\underbrace{\sum_{k=0}^{\infty} \Pr(s_0 \rightarrow s, k, \pi)}_{\text{\# of time steps spent on}} \right) \sum_a \nabla \pi(a|x) q_\pi(x, a) \\
 &= \sum_s \eta(s) \sum_a \nabla \pi(a|x) q_\pi(x, a)
 \end{aligned}$$

Next, we multiply everything by $\sum_{s'} \eta(s') / \sum_{s'} \eta(s')$

$$= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|x) q_\pi(x, a)$$

The Policy Gradient Theorem – Proof

[For simplicity, we leave it implicit that π is a function of θ , and all gradients are also implicitly with respect to θ .]

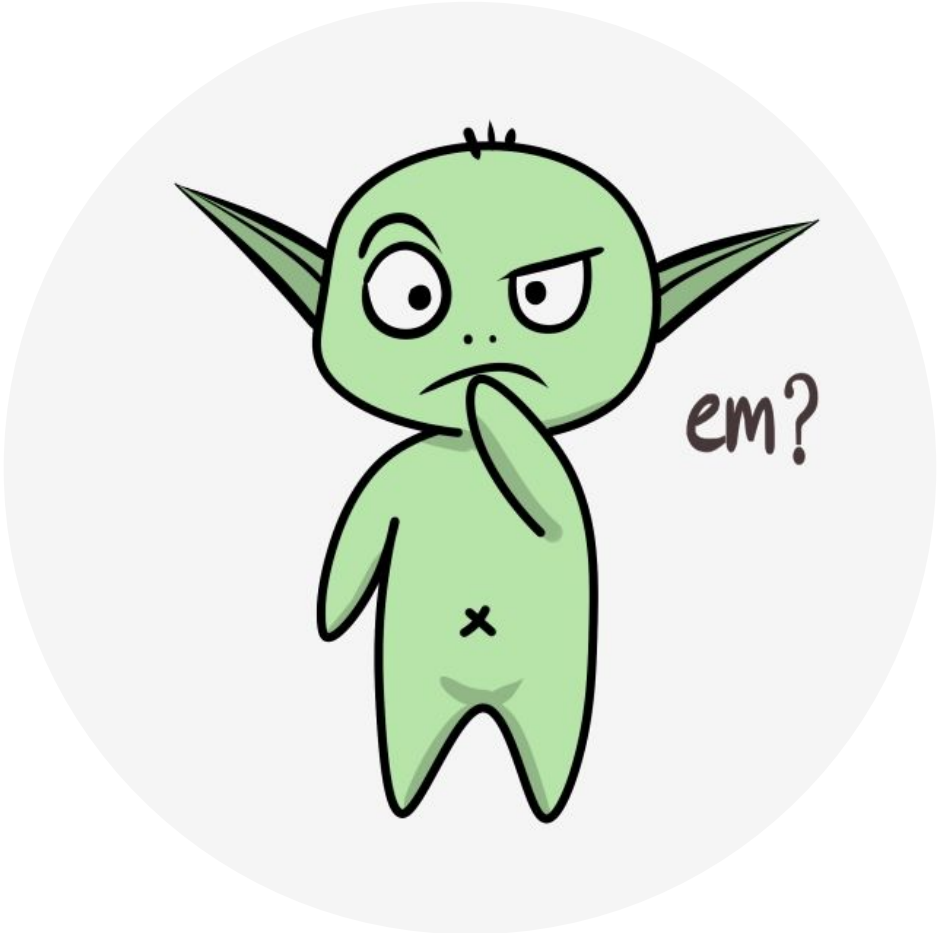
$$\begin{aligned}\nabla J(\theta) &= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|x) q_\pi(x, a) \\ &= \sum_{s'} \eta(s') \underbrace{\sum_s \mu(s)}_{=1} \sum_a \nabla \pi(a|x) q_\pi(x, a) \\ &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|x) q_\pi(x, a)\end{aligned}$$

Q.E.D.

The Policy Gradient Theorem

- The Policy Gradient Theorem [Marbach and Tsitsiklis, 1998, 2001; Sutton et al. 2000] provides an analytic expression for the gradient of performance w.r.t. the policy parameter that does not involve the derivative of the state distribution.

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta})$$



A First Policy Gradient Method

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta})$$

Any constant of proportionality can be absorbed into the step size α

$$= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right]$$

It weighs the sum by how often the states occur under the target policy π

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \mathbf{w}) \nabla \pi(a|S_t, \boldsymbol{\theta})$$

REINFORCE: Monte Carlo Policy Gradient [Williams, 1992]

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \mathbf{w}) \nabla \pi(a|S_t, \boldsymbol{\theta})$$

We want an update that at time t involves just A_t . We need to replace a sum over the RV's possible values by an expectation under π , and then sampling the expectation.

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_{\pi} \left[\sum_a q_{\pi}(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right] \\ &= \mathbb{E}_{\pi} \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_{\pi}(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right] \\ &= \mathbb{E}_{\pi} \left[q_{\pi}(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] = \mathbb{E}_{\pi} \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \end{aligned}$$

REINFORCE: Monte Carlo Policy Gradient [Williams, 1992]

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi} \left[G_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta})}{\pi(A_t | S_t, \boldsymbol{\theta})} \right]$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}$$

REINFORCE uses the full return,
thus it is a Monte Carlo method.

REINFORCE: Monte Carlo Policy Gradient [Williams, 1992]

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta})$$

Recall:

$$\nabla \ln x = \frac{\nabla x}{x}$$

Policy Gradient with Baseline

- As before, in the gradient bandits algorithm, we can generalize the policy gradient theorem to include a comparison of q_π to an arbitrary baseline $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s, a) - b(s) \right) \nabla \pi(a|s, \boldsymbol{\theta}).$$

- The baseline can be any function, even a random variable, as long as it does not vary with a ; because the subtracted quantity is zero (as before):

$$\sum_a b(s) \nabla \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0$$

REINFORCE with Baseline [Williams, 1992]

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \left(G_t - b(S_t) \right) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}$$

REINFORCE with Baseline [Williams, 1992]

REINFORCE with Baseline (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

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

$$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$$

$$\theta \leftarrow \theta + \alpha^{\theta} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \theta)$$



Actor-Critic Methods

- In REINFORCE with baseline, the learned state-value function estimates the value of the first state of each state transition.
- In actor-critic methods, the state-value function is applied also to the second state of the transition.
- When the state-value function is used to assess actions in this way it is called a critic, and the overall policy-gradient method is termed an actor–critic method.

One-Step Actor-Critic

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}.\end{aligned}$$

One-Step Actor-Critic

One-step Actor–Critic (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Initialize S (first state of episode)

$I \leftarrow 1$

 Loop while S is not terminal (for each time step):

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

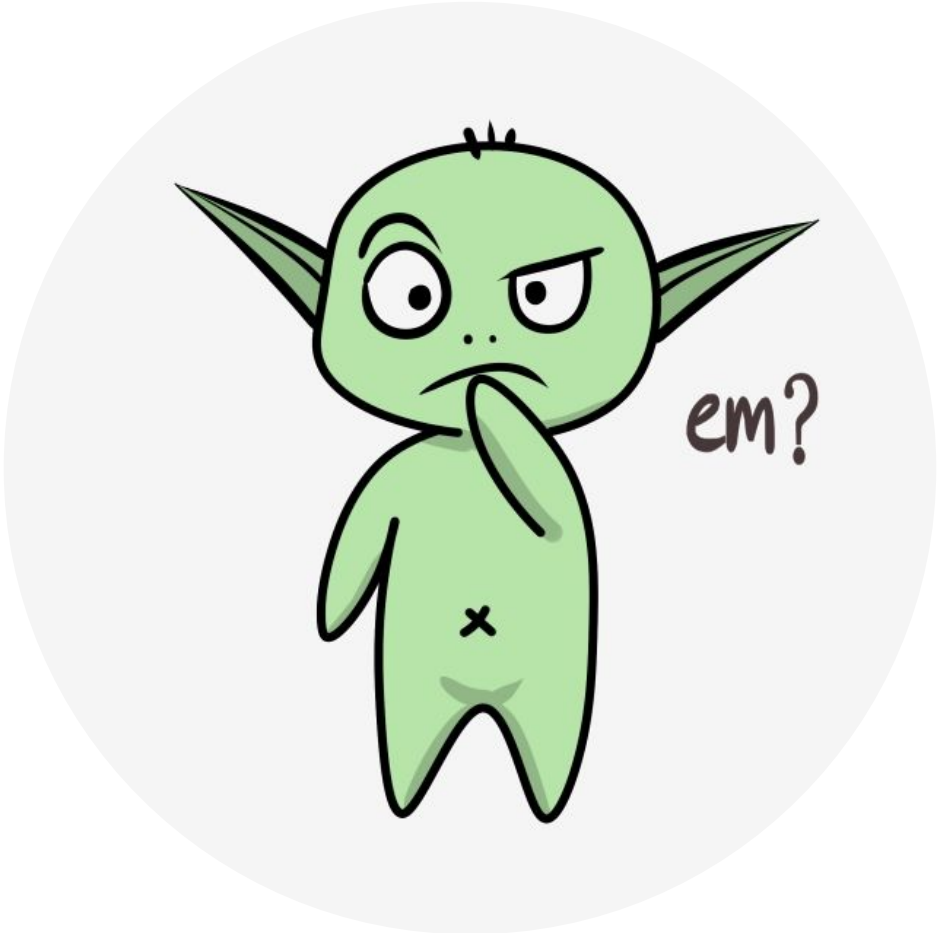
$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$

$\theta \leftarrow \theta + \alpha^{\theta} I \delta \nabla \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

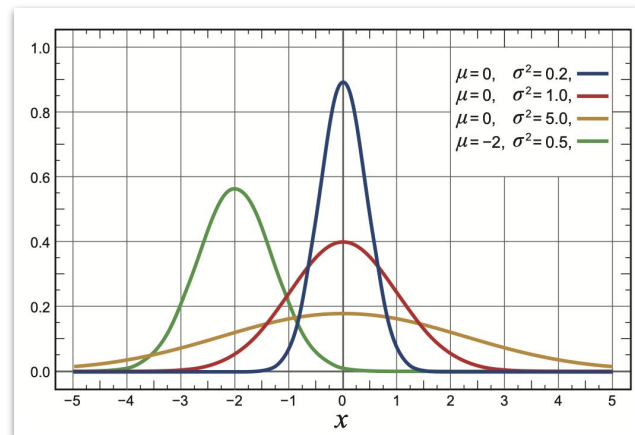
$S \leftarrow S'$



Policy Parameterization for Continuous Actions

- But how do we deal with large action spaces?
- We learn statistics of the probability distribution over actions to take.
 - E.g., the action set might be the real numbers, with actions chosen from a normal distribution.

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

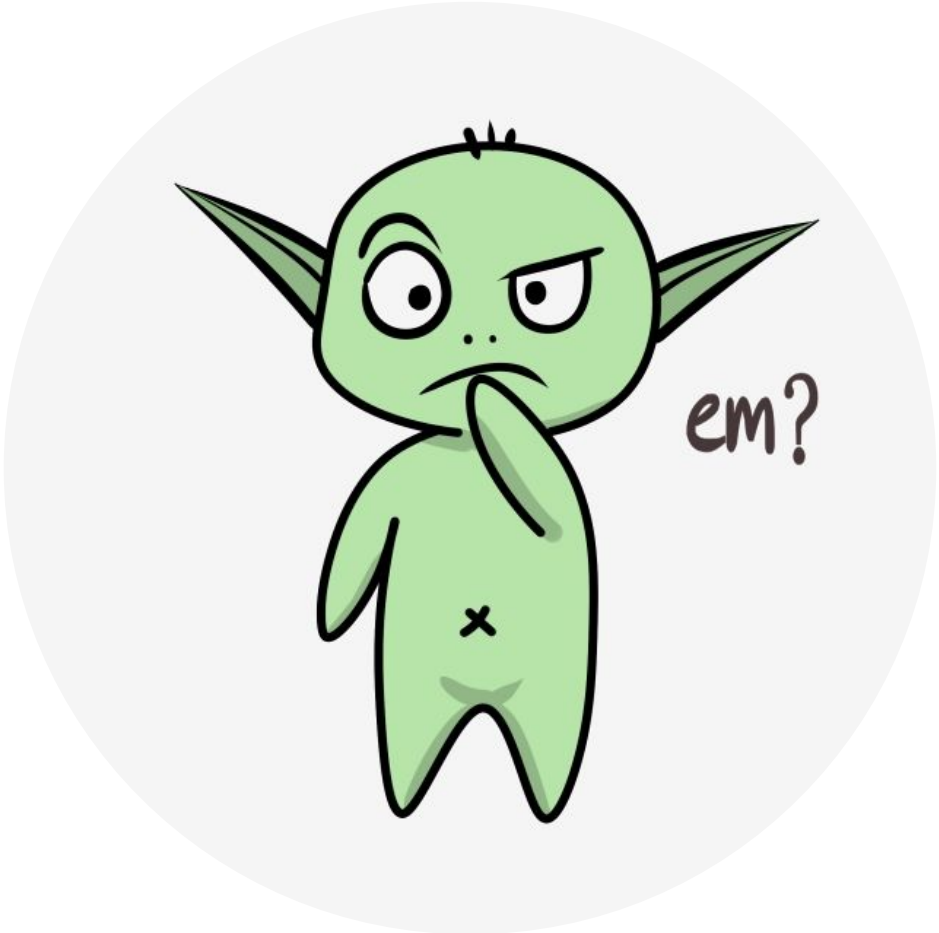


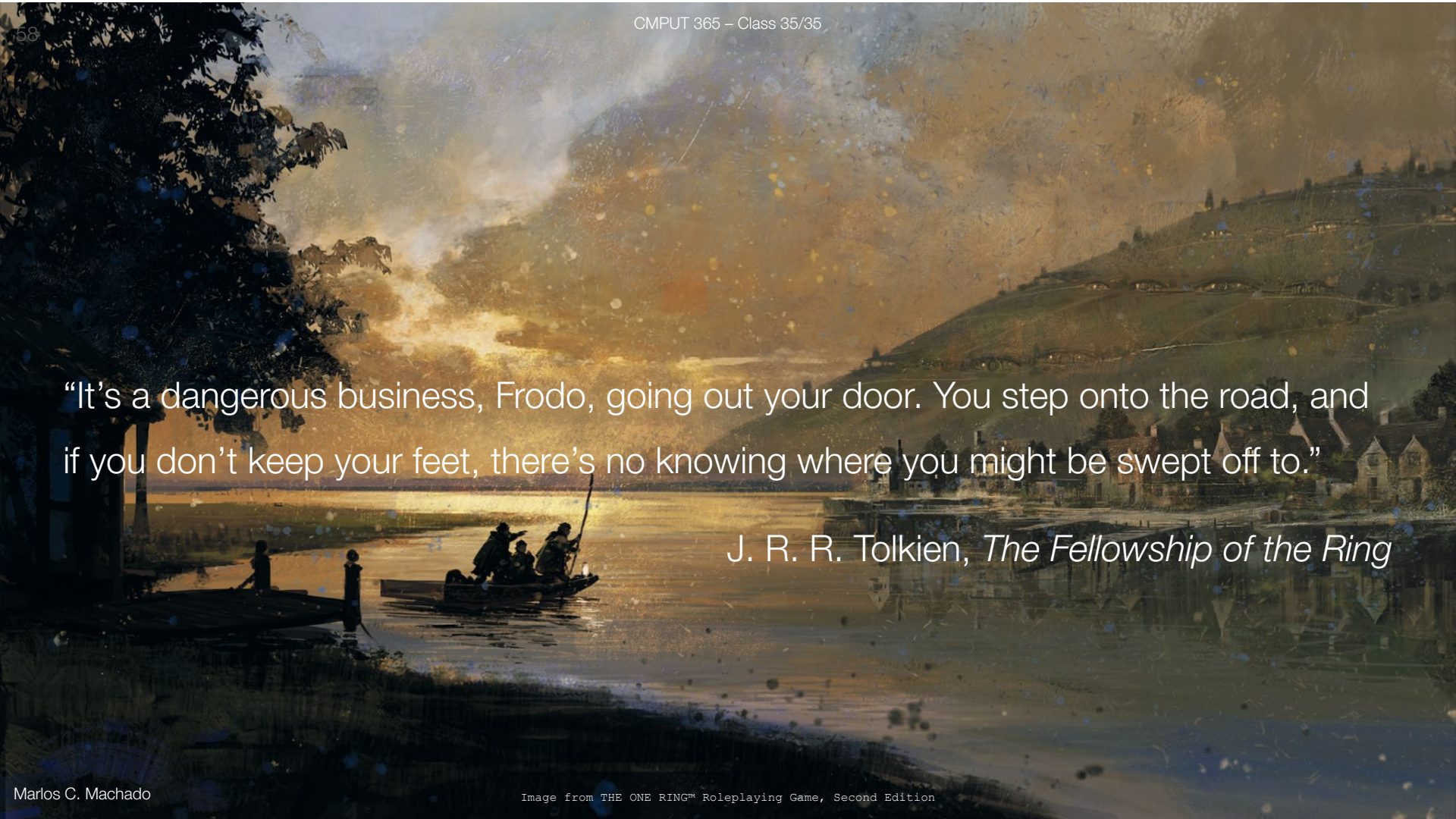
Policy Parameterization for Continuous Actions

- We need a policy parameterization, as always $\pi(a|s, \theta)$

$$\pi(a|s, \theta) \doteq \frac{1}{\sigma(s, \theta)\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \theta))^2}{2\sigma(s, \theta)^2}\right)$$

$$\mu(s, \theta) \doteq \theta_\mu^\top \mathbf{x}_\mu(s) \quad \text{and} \quad \sigma(s, \theta) \doteq \exp\left(\theta_\sigma^\top \mathbf{x}_\sigma(s)\right)$$





“It’s a dangerous business, Frodo, going out your door. You step onto the road, and if you don’t keep your feet, there’s no knowing where you might be swept off to.”

J. R. R. Tolkien, *The Fellowship of the Ring*

What's next?

Undergraduate courses:

- CMPUT 366 Search and Planning
- CMPUT 455 Search, Knowledge and Simulation
- CMPUT 466 Machine Learning Essentials
- CMPUT 467 Machine Learning II

Grad school 😊

- CMPUT 656 Human-in-the-Loop Reinforcement Learning by M. E. Taylor
- CMPUT 628 Deep Reinforcement Learning by M. C. Machado
- CMPUT 609 Reinforcement Learning II by R. Sutton
- CMPUT 653 Theoretical Foundations of Reinforcement Learning by C. Szepesvari
- CMPUT 653 Real-Time Policy Learning by A. R. Mahmood





“I am glad you are here with me. Here at the end of all things, Sam.”

J. R. R. Tolkien, *The Return of the King*