
“The test of a man isn’t what you think he’ll do. It’s what he actually does.”
Frank Herbert, *Dune*
CMPUT 628
Deep RL
Marlos C. Machado
https://pbs.twimg.com/media/FGHKuYcXwAA31Zx.jpg
Classes 16–18/ 25

Reminders & Notes

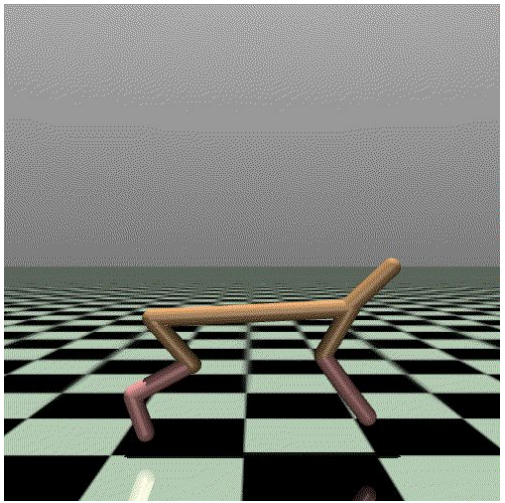
- I extended the due date of Assignment 3 to Friday, 6.
- Assignment 4 is due on March 14
- Seminars start in 3 weeks-ish

Please, interrupt me at any time!

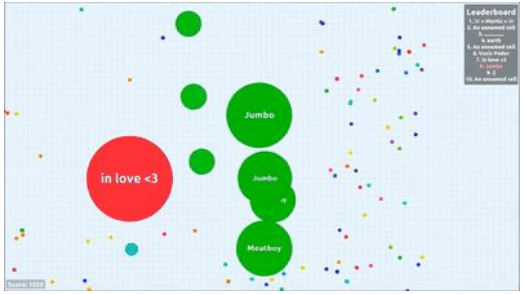
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

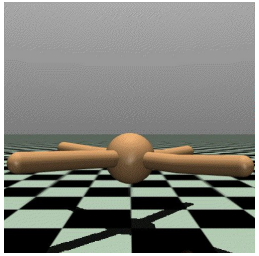
Lots of problems can be seen as *continuous* RL problems

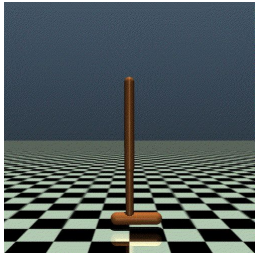
- Pretty much everything we discussed so far estimates some version of the action-value function, Q(s,a), and selects actions based on it, e.g.; argmax_a Q(s,a)
- Taking the max over 20 actions is fine, how do we go about ∞?
  - Obviously, taking the max is tricky, but there's also the neural network architecture; we can't have an infinite number of heads to estimate Q(s,a) for all a $\in \mathcal{A}$
- Discretizing actions also doesn't solve anything
  - The observation space in *half-cheetah* consists of 9 links and 8 joints
  - Actions consist of applying torque over the thighs, shins, and feet (6)
  - Even if we discretize the actions to be only -1, 0, and 1; we would still have $3^6 = 729$ actions

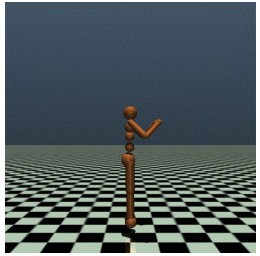


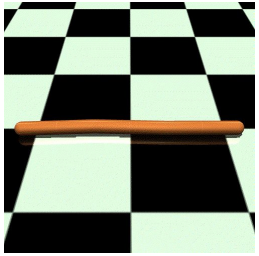
Many problems have continuous action spaces





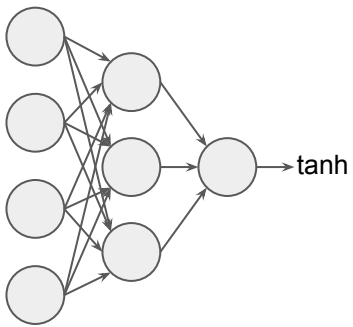






Can we adapt DQN to tackle continuous actions spaces?

- Again, taking the max is tricky, and even the neural network architecture itself would need to be adapted, since we can't have independent heads per action
- The solution? What if instead we directly output the action to be taken?



- We are directly parameterizing the policy (so it is a policy gradient method), but we are doing so with a deterministic policy
- But how do we update the policy parameters?!

Refresher: Policy gradient methods

Marlos C. Machado

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):

At first, all action preferences are the same

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

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

Some more on the Softmax

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

	a_1	a_2	a_3
H(·)	1	1	1
π(·)	0.33	0.33	0.33

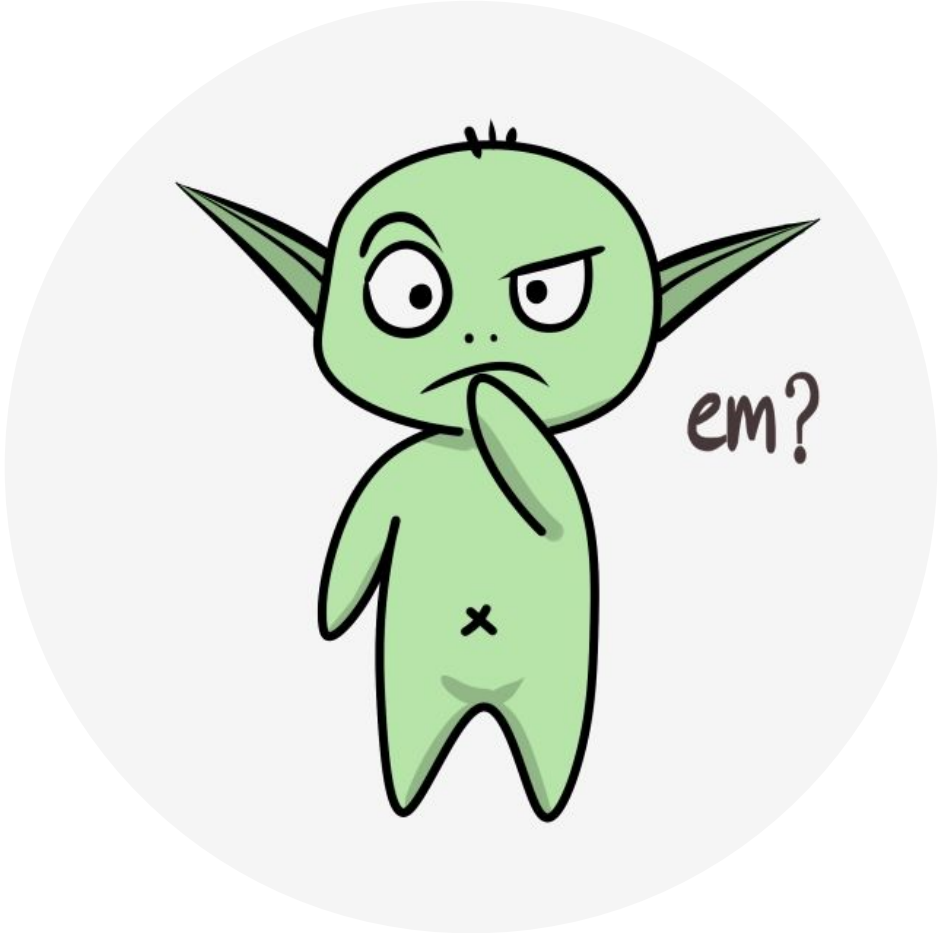
	a_1	a_2	a_3
H(·)	4	1	1
π(·)	0.91	0.05	0.05

	a_1	a_2	a_3
H(·)	3	2	1
π(·)	0.67	0.24	0.09

	a_1	a_2	a_3
H(·)	3	1	1
π(·)	0.79	0.11	0.11

	a_1	a_2	a_3
H(·)	-4	1	1
π(·)	0	0.5	0.5

	a_1	a_2	a_3
H(·)	4	3	2
π(·)	0.67	0.24	0.09



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

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:

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

$$= \sum_x q_*(x) \frac{\partial \pi_t(s)}{\partial H_t(a)}$$

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

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

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

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

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

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

Quotient rule:

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

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

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

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

Gradient Bandit Algorithms (Chapter 2) – Derivation

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

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

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.

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

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

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

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

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

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

Gradient Bandit Algorithms (Chapter 2) – Baseline

Let's look at the exact performance gradient:

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

Instead, we could do:

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

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 $\pi_t(x)/\pi_t(x)$:

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

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

Gradient Bandit Algorithms (Chapter 2) – No parameterization

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

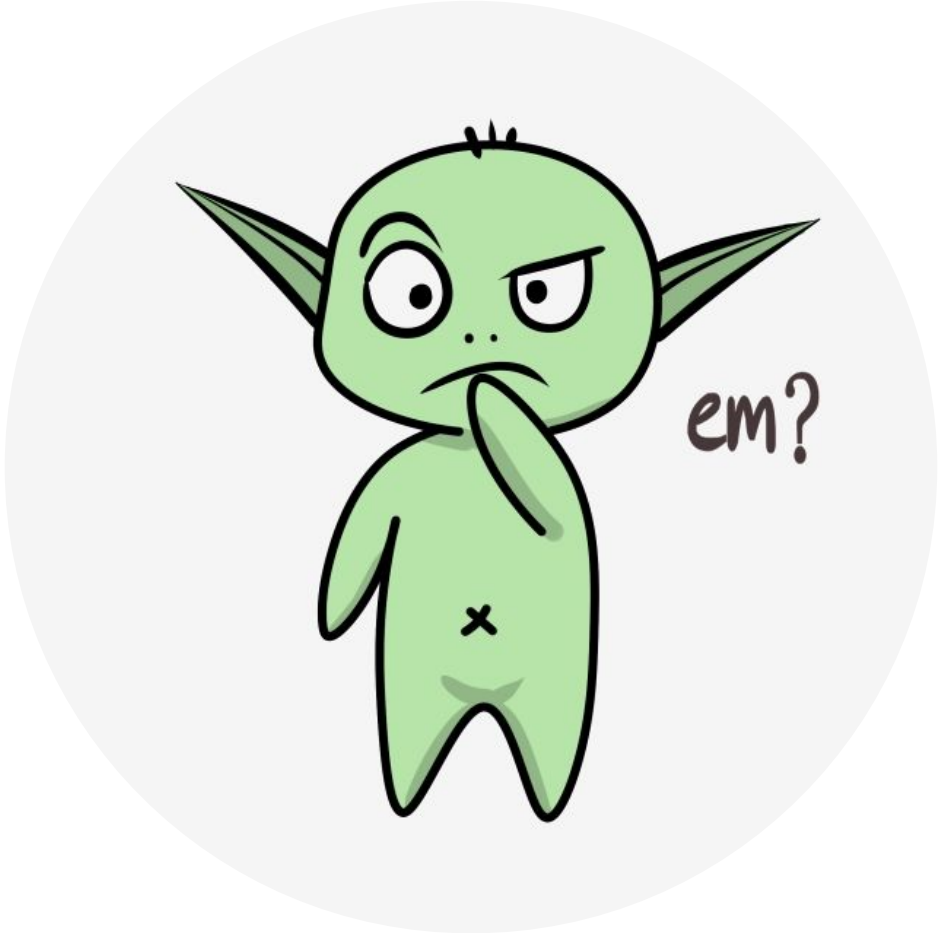
$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} \qquad \text{where } \mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

with our derivation, we then have:

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

Which means:

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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

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(**θ**):

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

- 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

A First Policy Gradient Method

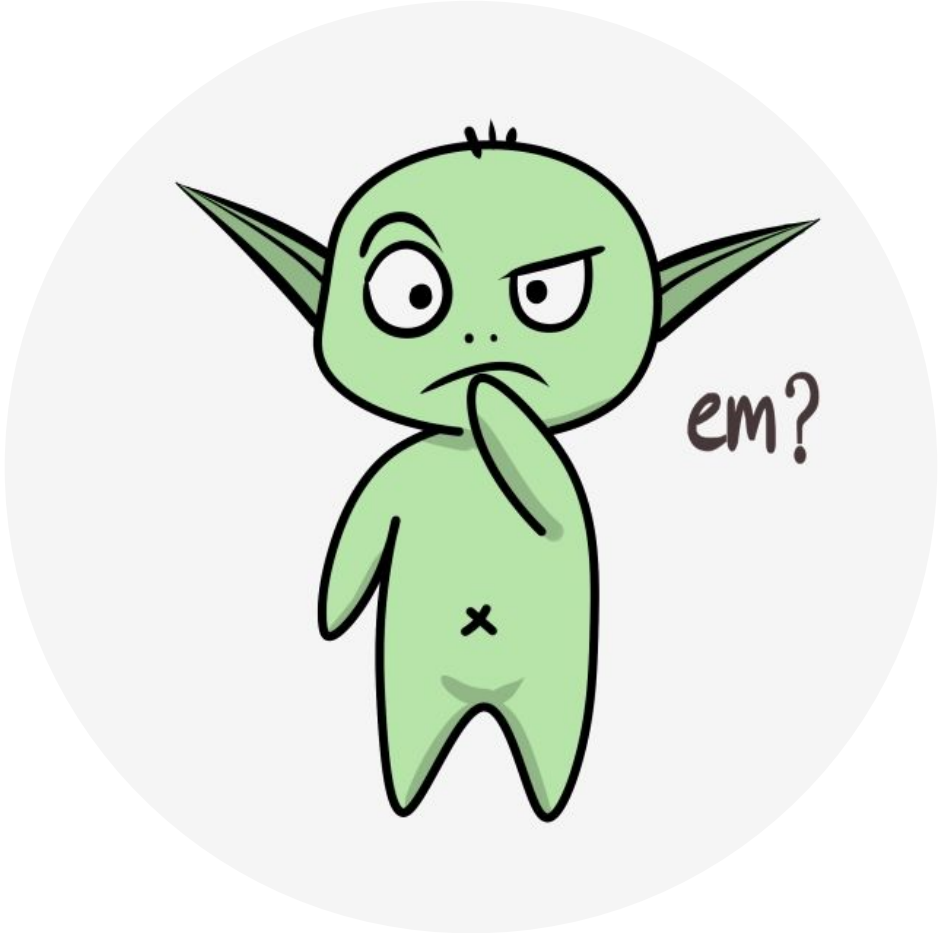
$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$

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

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

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

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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

REINFORCE: Monte Carlo Policy Gradient [Willliams, 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.

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

REINFORCE: Monte Carlo Policy Gradient [Willliams, 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 [Willliams, 1992]

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

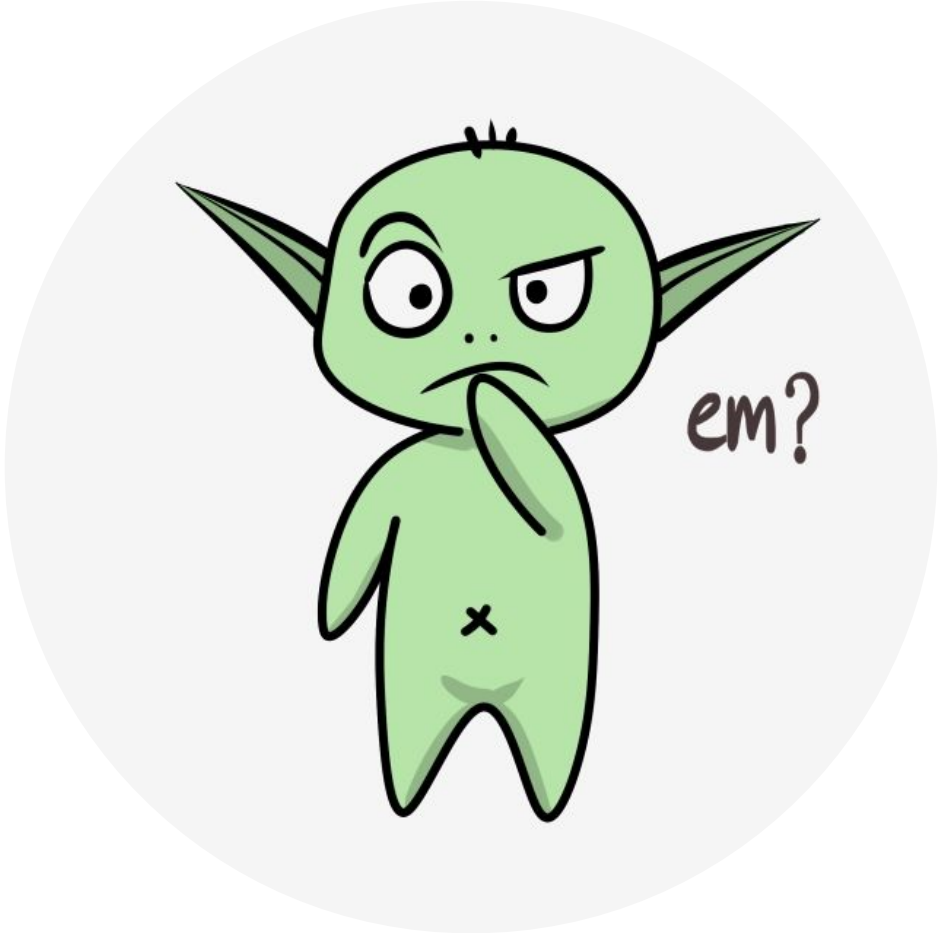
REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

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 $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \boxed{\ln \pi(A_t|S_t,\boldsymbol{\theta})}$

Recall:

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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

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.

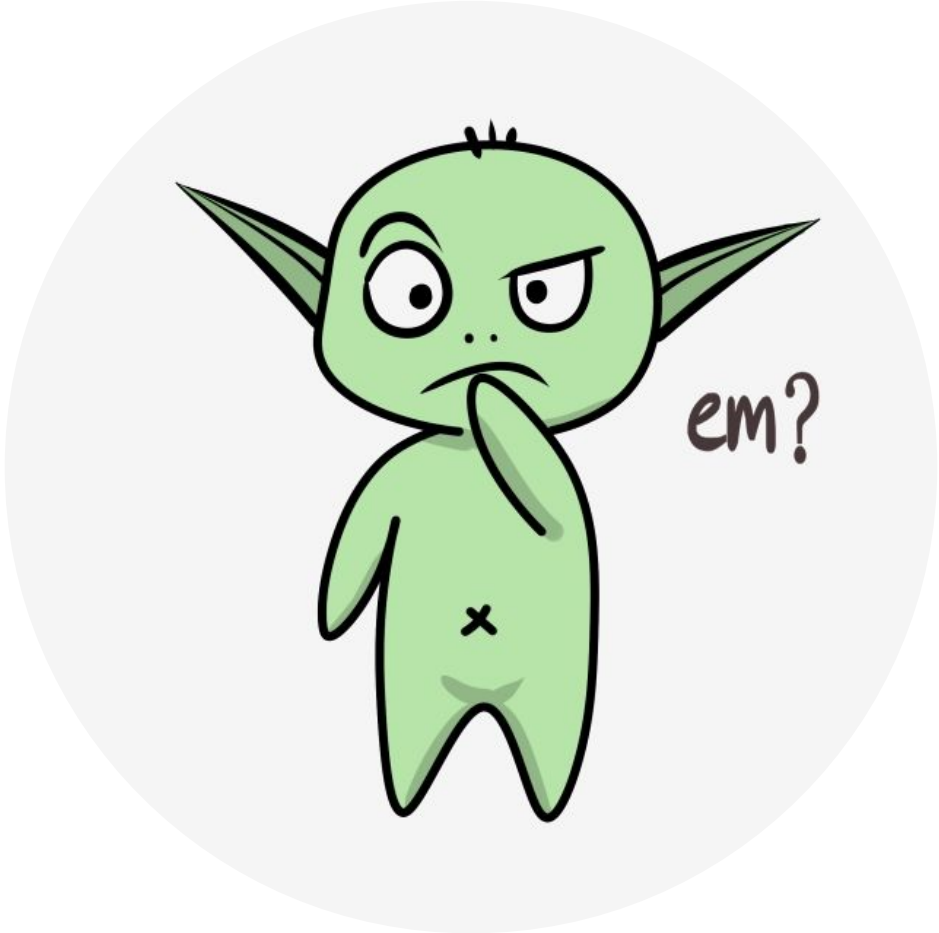
One-Step Actor-Critic

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

One-Step Actor-Critic

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

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ 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):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$



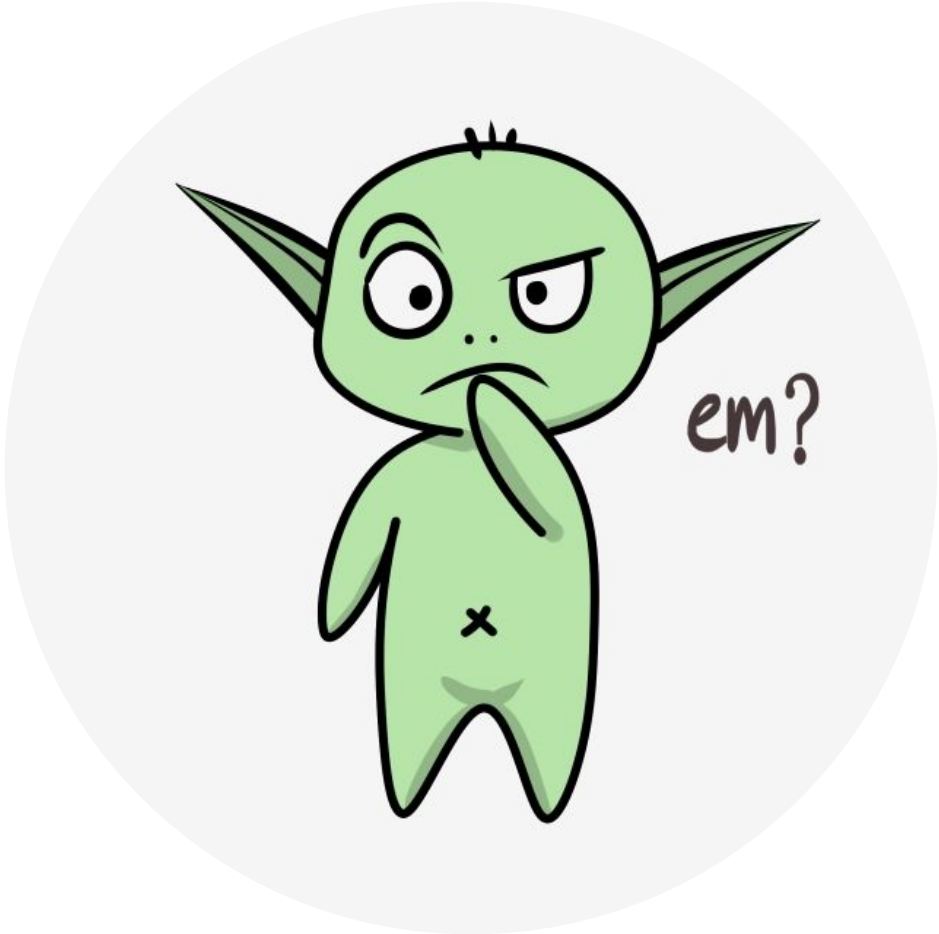
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Actor-Critic Methods

- Policy gradient theorem: $\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$
- … but how do we estimate q_π? REINFORCE: $\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$
- Monte Carlo methods have high-variance, they are slow. What if we used TD?

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

… that's an actor-critic method!



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Deterministic Policy Gradient [Silver et al., 2014]

- A “regular” policy gradient method, with stochastic policies, can be seen as sampling an action and if the return is high, it increases probability of that action.
- DPG instead, moves the action in the direction that increases Q. So, instead of changing probabilities it directly changes actions, not having to wait for the “right” action to be sampled (which, in the continuous case, it never will).
- There is no free-lunch, though:
  - Instead of sampling points on a surface and adjusting the prob. mass toward high regions, DPG takes the grad. of the surface and moves directly uphill. If your surface is wrong, you are misled.
  - DPG trades off low variance by bias from a critic approximation (ML 101 again).

Deterministic Policy Gradient [Silver et al., 2014]

Theorem 1 (Deterministic Policy Gradient Theorem). *Suppose that the MDP satisfies conditions A.1 (see Appendix; these imply that $\nabla_\theta \mu_\theta(s)$ and $\nabla_a Q^\mu(s,a)$ exist and that the deterministic policy gradient exists. Then,*

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \left.\nabla_a Q^\mu(s,a)\right|_{a=\mu_\theta(s)} \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \left.\nabla_a Q^\mu(s,a)\right|_{a=\mu_\theta(s)}\right] \quad (9)$$

(improper) discounted state distribution

Deterministic Policy Gradient [Silver et al., 2014]

- An actor-critic method (model-free) that learns a *deterministic* policy (actor) updated by the action-value function estimates from the critic
  - There have been concerns about whether one could get the policy gradient theorem for deterministic policies; this is beyond of the scope of this class, but see discussion by Silver et al. (2014) if you want to know more

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \; \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}$$

  - This update is on-policy, it only works for didactic purposes; how are we going to explore?

Deterministic Policy Gradient [Silver et al., 2014]

- Off-policy actor-critic algorithm:

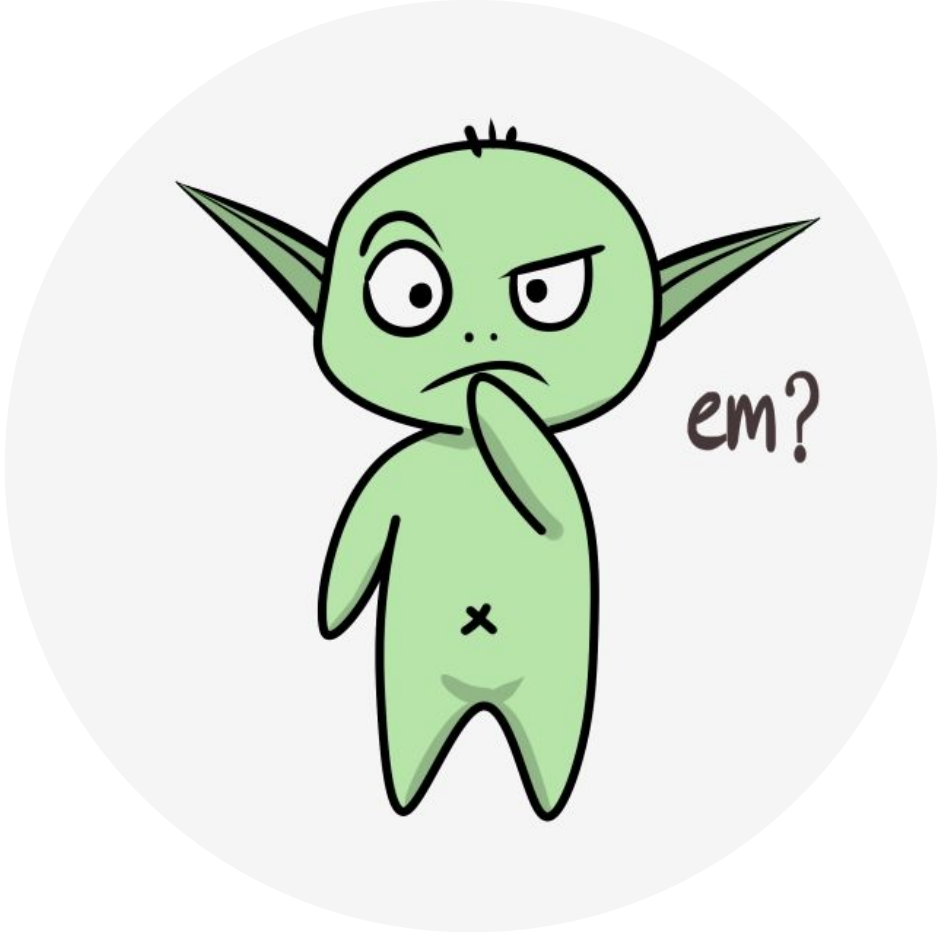
$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \; \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}$$

  - Notice that, in the stochastic case, the PG theorem is $\nabla_\theta J(\theta)=E_{s\sim\rho\pi,a\sim\pi\theta(\cdot|s)}[\nabla_\theta \log\pi_\theta(a|s) q_\pi(s,a)]$, and in the deterministic case it is $\nabla_\theta J(\theta)=E_{s\sim\rho\pi}[\nabla_\theta \pi_\theta(s) \nabla_a Q_\pi(s,a) \mid_{a=\pi\theta(s)}]$. The difference here is that in the stochastic case, we are conditioning the gradient on sampled actions, while in the deterministic case we are evaluating the actions directly produced by the deterministic policy. This means the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient since we do not need to sample actions from the log probability.

Why no importance sampling? [Silver et al., 2014]

- “The deterministic policy gradient removes the integral over actions, we can avoid importance sampling in the actor; and by using Q-learning, we can avoid importance sampling in the critic.” [Silver et al., 2014]

$$\begin{aligned}\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \; \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}\end{aligned}$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Deep Deterministic Policy Gradient [Lilicrap et al., 2016]

- “Our contribution here is to provide modifications to DPG, inspired by the success of DQN, which allow it to use neural network function approximators to learn in large state and action spaces online”
- They leverage the key insights of DQN: target network and experience replay buffers; plus batch normalization (Ioffe & Szegedy, 2015), which was new at the time
  - Batch normalization allows for varied input ranges when dealing with proprioception inputs
- “A key feature of the approach is its simplicity: it requires only a straightforward actor-critic architecture and learning algorithm with very few ‘moving parts’, making it easy to implement and scale to more difficult problems and larger networks.”

Deep Deterministic Policy Gradient [Lilicrap et al., 2016]

- They use a Polyak averaging for the target network for both the actor and critic networks (yes, we have four networks now). “This means that the target values are constrained to change slowly, greatly improving the stability of learning”.
  - $\theta^- \leftarrow \tau\theta^- + (1 - \tau)\theta \quad \text{with} \quad \tau << 1$
  - “We found that having both a target μ’ and Q’ was required to have stable targets (...) in order to consistently train the critic without divergence”
- For exploration, they add noise to the chosen action: $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$
  - Lilicrap et al., 2016 proposed the use of “an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) to generate temporally correlated exploration for exploration”
  - “More recent results suggest that uncorrelated, mean-zero Gaussian noise works perfectly well” (OpenAI, https://spinningup.openai.com/en/latest/algorithms/ddpg.html)

DDPG

[Lillicrap et al., 2016]

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

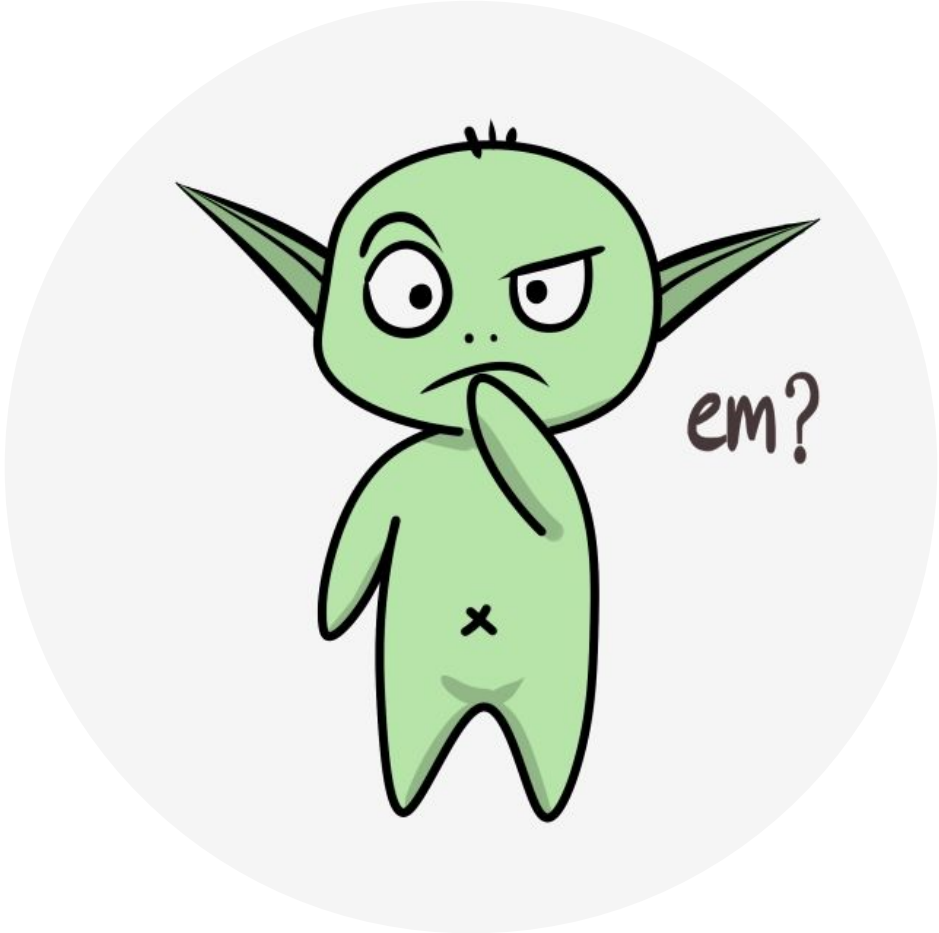
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

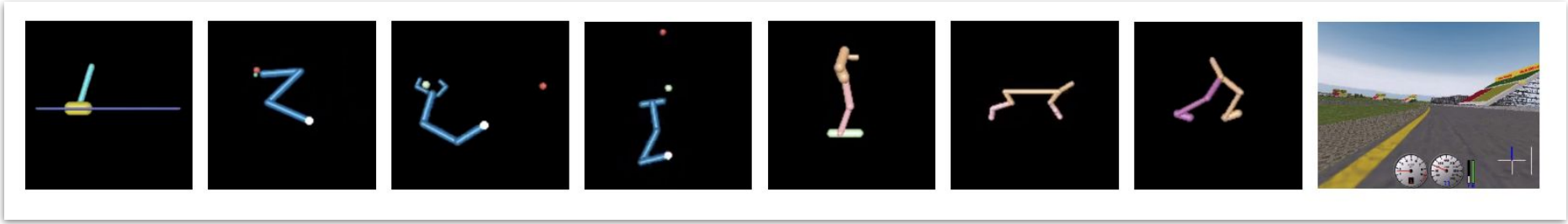
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

DDPG results in MuJoCo tasks (and TORCS) [Lilicrap et al., 2016]



- They also used a frameskip (and framestacking) of 3
- They report evaluation results without exploration noise
- They report results with inputs as pixels and proprioception
  - For proprioception, they used an MLP with two hidden layers
  - For pixel inputs, they used a 3-layer convolutional network followed by two hidden layers

DDPG [Lilicrap et al., 2016]

Table 1: Performance after training across all environments for at most 2.5 million steps. We report both the average and best observed (across 5 runs). All scores, except Torcs, are normalized so that a random agent receives 0 and a planning algorithm 1; for Torcs we present the raw reward score. We include results from the DDPG algorithn in the low-dimensional ($lowd$) version of the environment and high-dimensional (pix). For comparision we also include results from the original DPG algorithm with a replay buffer and batch normalization ($cntrl$).

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Ablations on DDPG [Lilicrap et al., 2016]

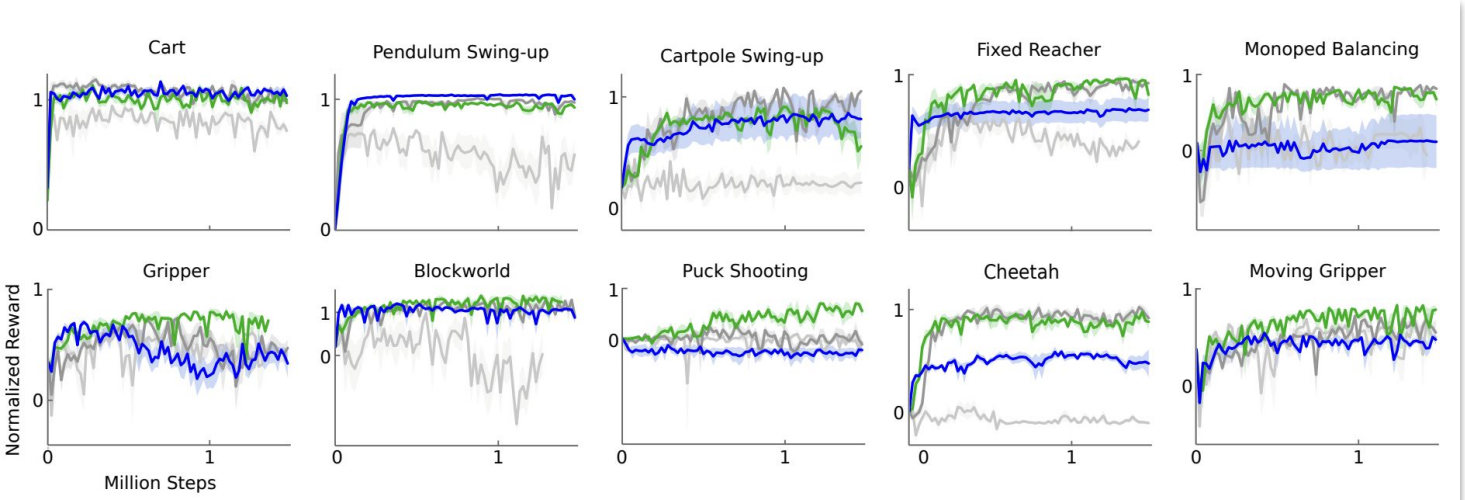
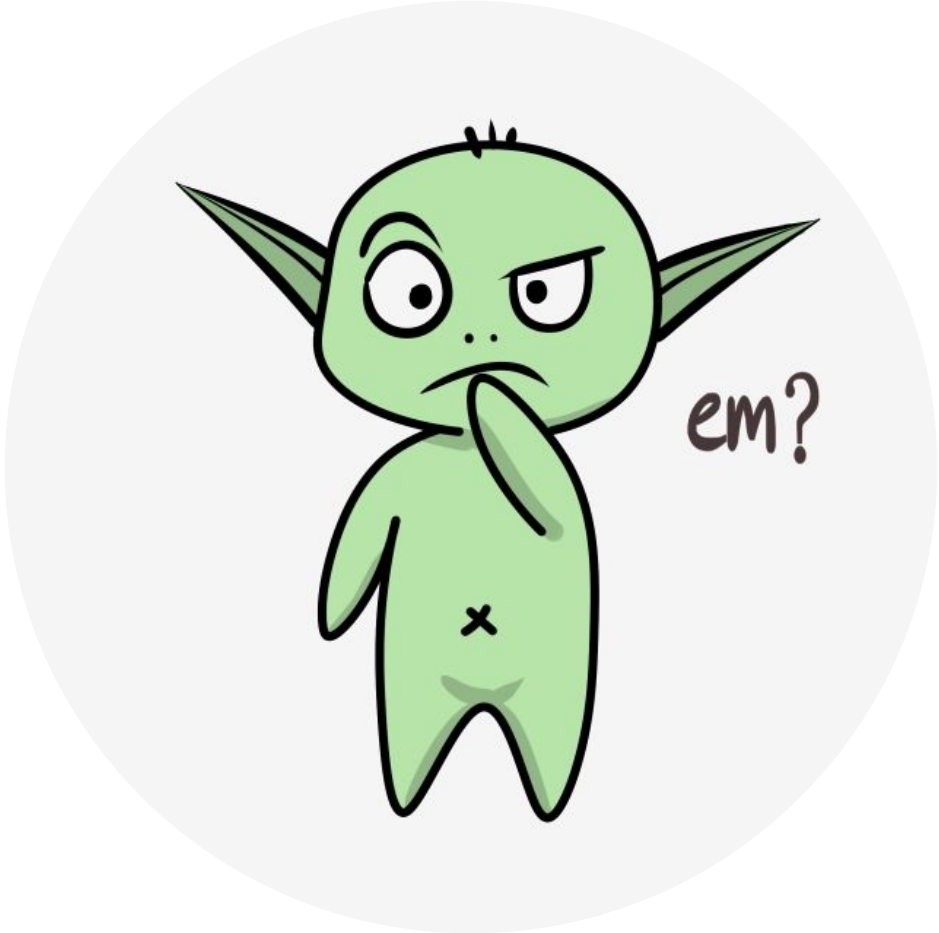


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

In fact, we can also do distributed computing with DDPG

[Horgan et al., 2018]

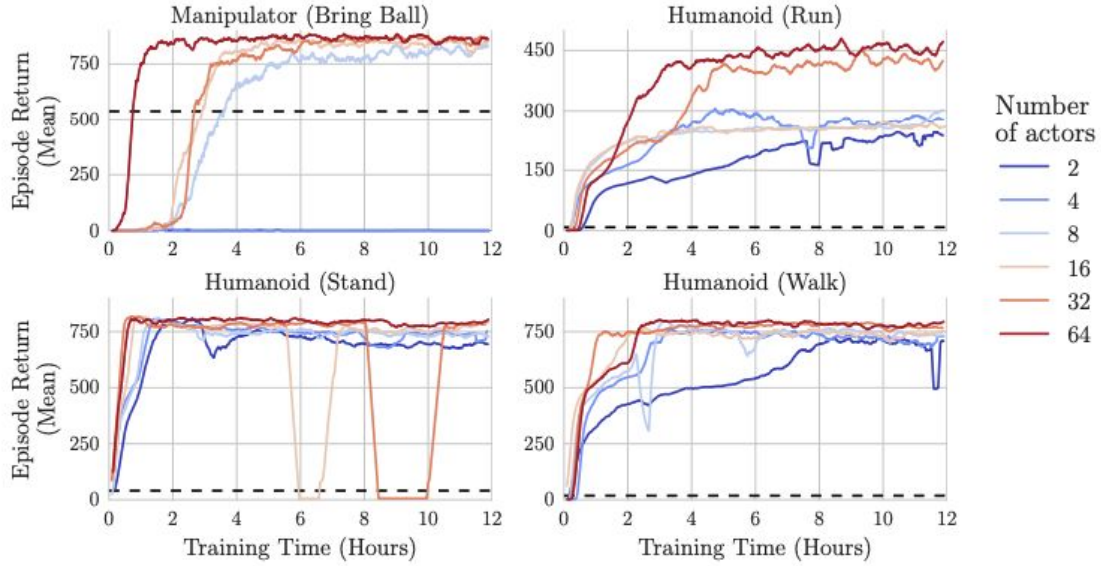
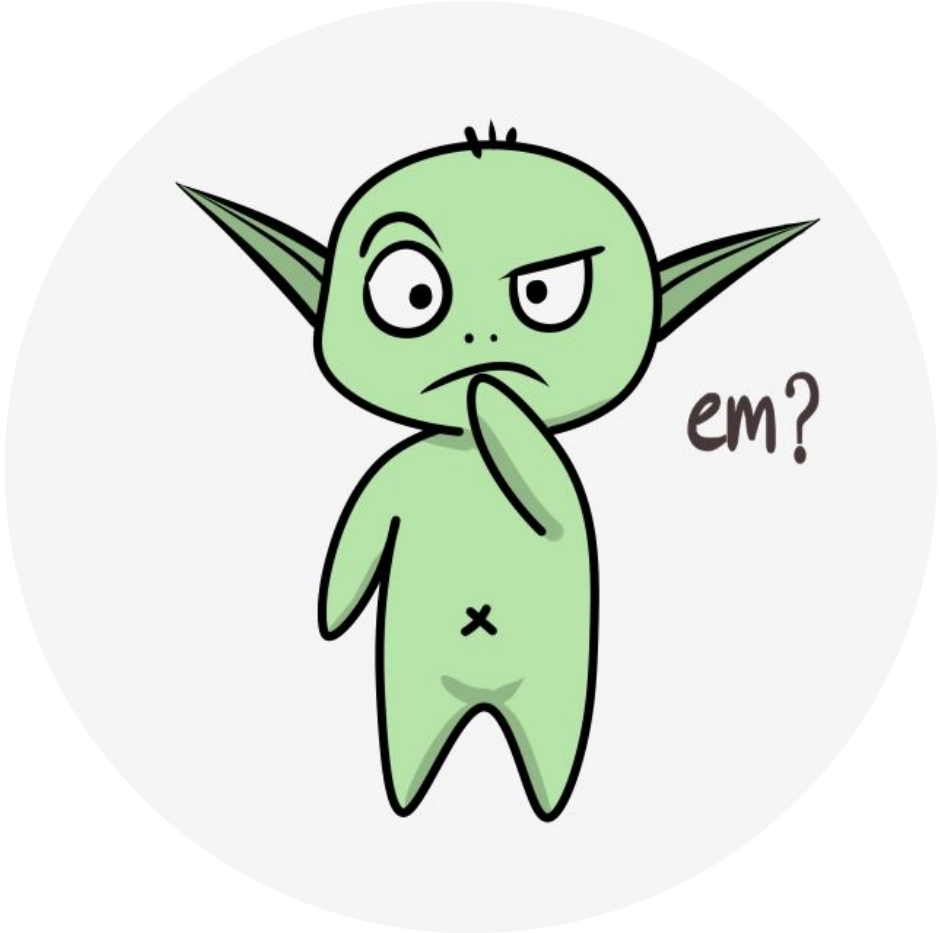


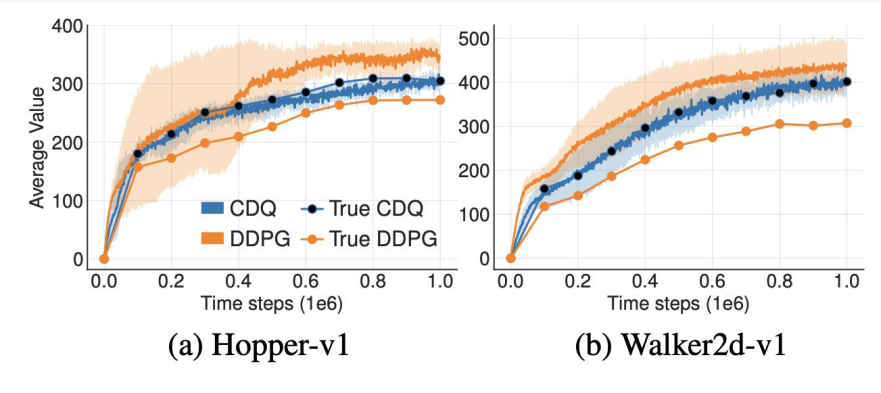
Figure 3: Performance of Ape-X DPG on four continuous control tasks, as a function of wall clock time. Performance improves as we increase the numbers of actors. The black dashed line indicates the maximum performance reached by a standard DDPG baseline over 5 days of training.

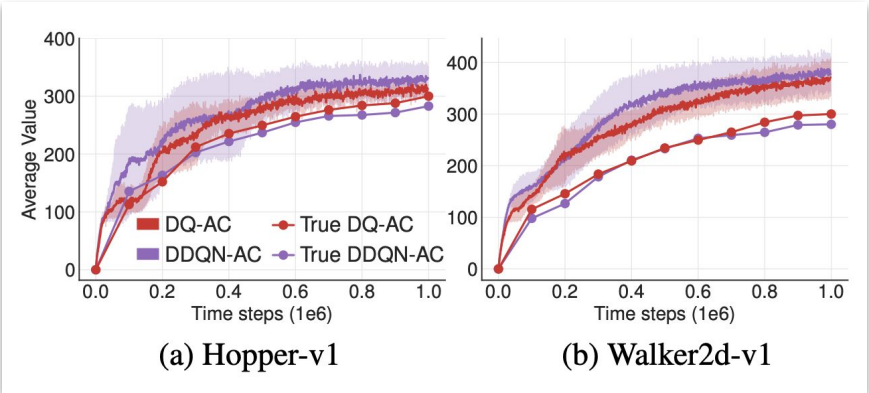


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

A Better DDPG

- DDPG can be quite sensitive to hyperparameter choices
- Apparently, DDPG can greatly overestimate the action values (remember double learning?)
- Fujimoto et al. (2018) show that overestimation impacts actor-critic methods too
- TD3 is an algorithm that tries to reduce maximization bias and variance.





The 3 New Components of Twin Delayed DDPG (TD3)

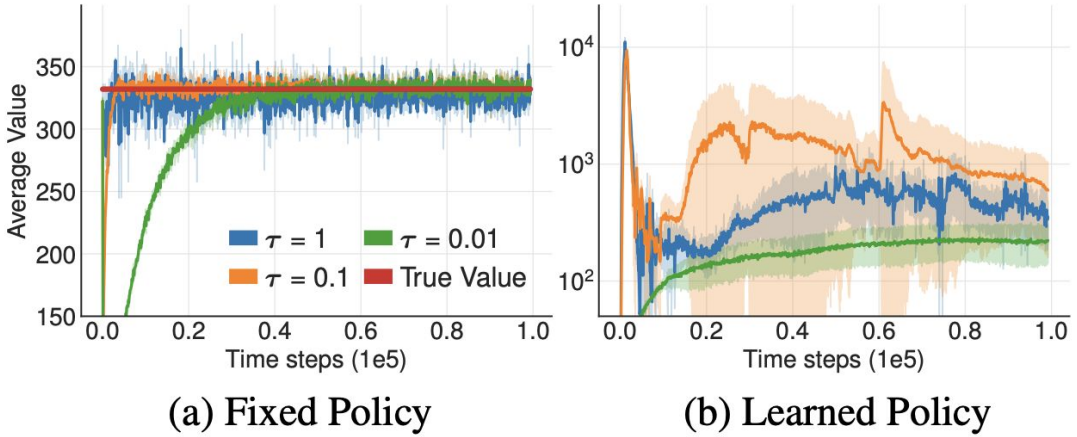
[Fujimoto et al., 2018]

- *Clipped Double-Q Learning*: Learn two Q functions and it sets the minimum between the two to bootstrap from when defining the target to regress to
  ↑ This is where "*Twin"* comes from
- "Delayed" Policy Updates: The actor (policy) and the target network are updated less frequently than the critic's Q-function (to get better predictions first)
  ↑ This is where *"Delayed"* comes from
- Target Policy Smoothing: We add noise to the target action as a type of implicit regularization – The idea is that actions nearby have similar values, avoiding "sharp peaks"

TD3 is very focused on variance reduction, they show the target network also helps

Stabilizing Learning with Target Networks

[Fujimoto et al., 2018]



(a) Fixed Policy (b) Learned Policy

Figure 3. Average estimated value of a randomly selected state on Hopper-v1 without target networks, ($\tau = 1$), and with slow-updating target networks, ($\tau = 0.1, 0.01$), with a fixed and a learned policy.

Target Policy Smoothing in TD3 [Fujimoto et al., 2018]

- We regress to the target with some noise added to it, to smooth things out
  - Otherwise we can have some sharp peaks that are undesirable
  - “Similar actions should have similar values”

$$Y_{TD3}(O_{t+1}, R_{t+1}; \boldsymbol{\phi}^-, \boldsymbol{\theta}^-) = R_{t+1} + \gamma Q\Big(O_{t+1}, \pi(O_{t+1}; \boldsymbol{\phi}^-) + \epsilon; \boldsymbol{\theta}^-\Big)$$

$$\epsilon \sim \text{clip}\Big[\mathcal{N}(0, \sigma)\Big]_{-c}^{+c}$$

Reducing Overestimation: Clipped Double-Q Learning

[Fujimoto et al., 2018]

- We use the minimum between the two action-value estimates we have
- “Double DQN (Van Hasselt et al., 2016), to be ineffective in an actor-critic setting (...) Unfortunately, due to the slow-changing policy in an actor-critic setting, the current and target value estimates remain too similar to avoid maximization bias.”

$$Y^1_{TD3} = Y_{TD3}(O_{t+1}, R_{t+1}; \boldsymbol{\phi}_1^-, \boldsymbol{\theta}_1^-) = R_{t+1} + \gamma Q\Big(O_{t+1}, \pi(O_{t+1}; \boldsymbol{\phi}_1^-) + \epsilon; \boldsymbol{\theta}_1^-\Big)$$

$$Y_{TD3} = \min(Y^1_{TD3}, Y^2_{TD3})$$

$$\mathcal{L}_1^{\text{TD3}} = \mathbb{E}_{(o,a,r,o')\sim U(\mathcal{D})}\Big[\big(Y_{TD3} - Q(O_t, A_t; \boldsymbol{\theta}_{t,1})\big)^2\Big]$$

We do it for both networks

“Delayed” Policy Updates in TD3 [Fujimoto et al., 2018]

- As in DDPG, we just directly maximize the actor network; but we do so less frequently than the critic network
  - “If target networks can be used to reduce the error over multiple updates, and policy updates on high-error states cause divergent behavior, then the policy network should be updated at a lower frequency than the value network, to first minimize error before introducing a policy update.”
- Update the policy and target networks after a fixed number of updates *d* to the critic—Yup, another hyperparameter

$$\max_{\boldsymbol{\phi}} \mathbb{E}_{(o,a,r,o')\sim U(\mathcal{D})} \Big[Q\big(O_t, \pi(O_t;\boldsymbol{\phi});\boldsymbol{\theta}\big) \Big]$$

Guess what? It works ¯_(ツ)_/¯ [Fujimoto et al., 2018]

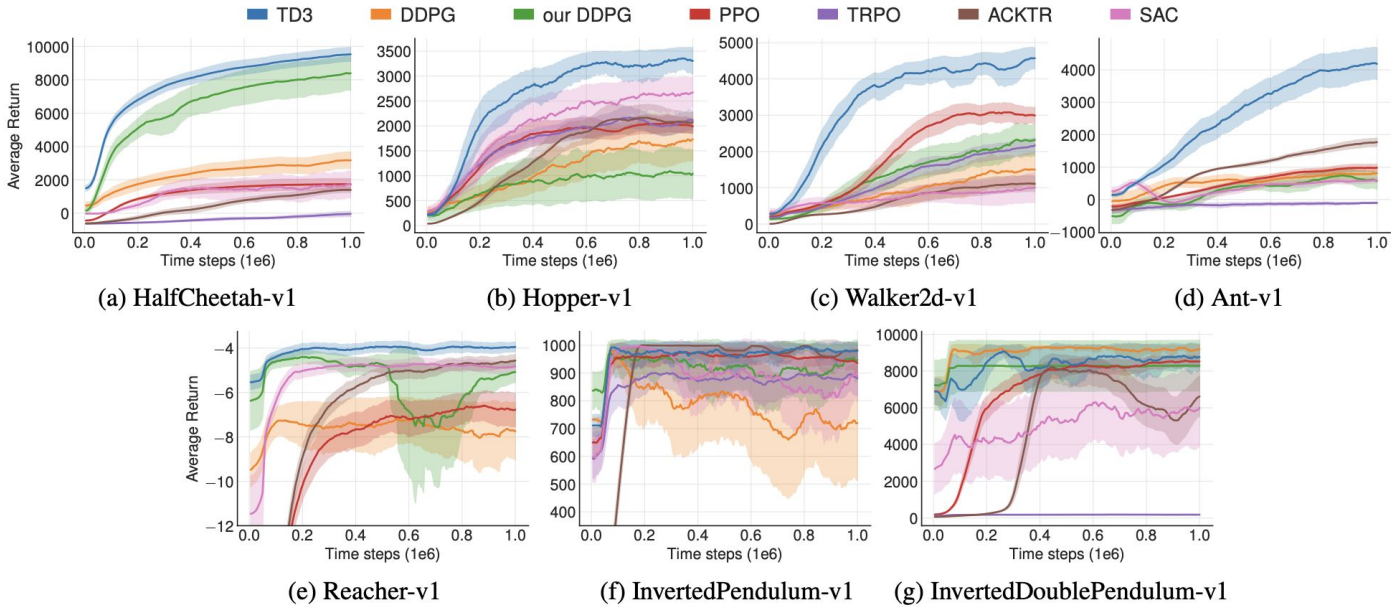


Figure 5. Learning curves for the OpenAI gym continuous control tasks. The shaded region represents half a standard deviation of the average evaluation over 10 trials. Curves are smoothed uniformly for visual clarity.

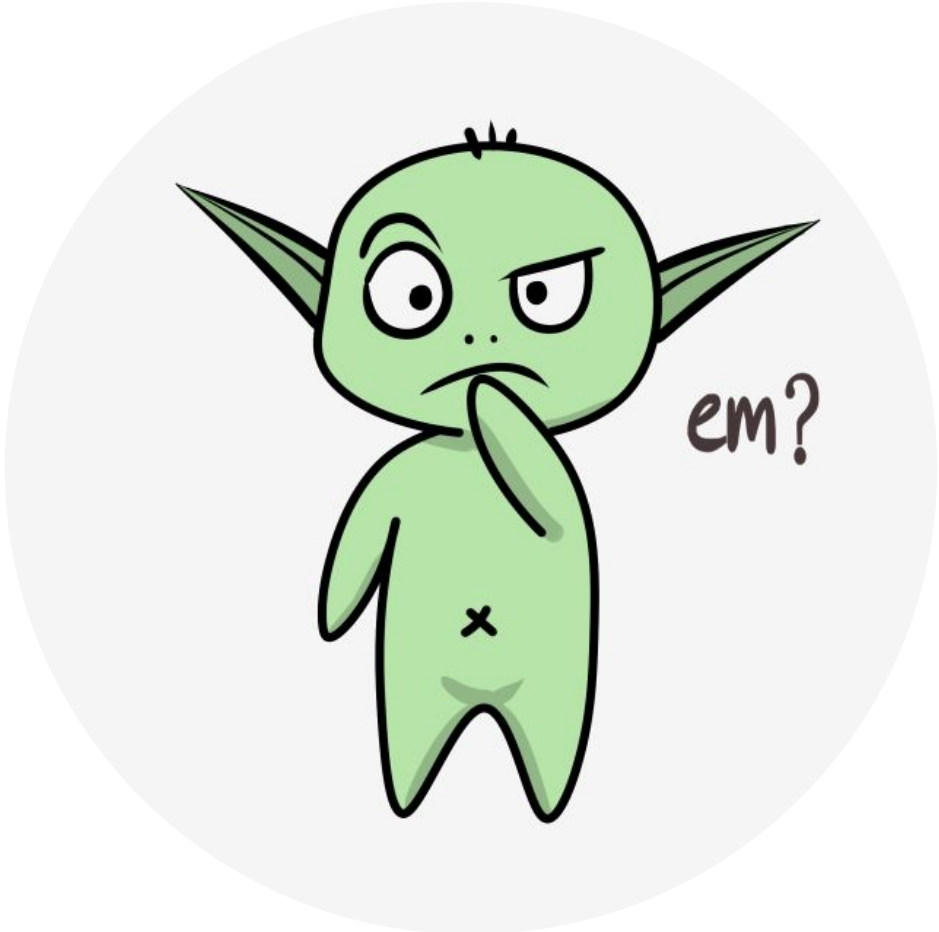
What about wall-clock-time?

And as usual, they have ablations [Fujimoto et al., 2018]

Method	HCheetah	Hopper	Walker2d	Ant
TD3	9532.99	**3304.75**	**4565.24**	**4185.06**
DDPG	3162.50	1731.94	1520.90	816.35
AHE	8401.02	1061.77	2362.13	564.07
AHE + DP	7588.64	1465.11	2459.53	896.13
AHE + TPS	9023.40	907.56	2961.36	872.17
AHE + CDQ	6470.20	1134.14	3979.21	3818.71
TD3 - DP	9590.65	2407.42	**4695.50**	3754.26
TD3 - TPS	8987.69	2392.59	4033.67	**4155.24**
TD3 - CDQ	9792.80	1837.32	2579.39	849.75
DQ-AC	9433.87	1773.71	3100.45	2445.97
DDQN-AC	**10306.90**	2155.75	3116.81	1092.18

Table 2. Average return over the last 10 evaluations over 10 trials of 1 million time steps, comparing ablation over delayed policy updates (DP), target policy smoothing (TPS), Clipped Double Q-learning (CDQ) and our architecture, hyper-parameters and exploration (AHE). Maximum value for each task is bolded.

A note on ablations…



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

What if we want a stochastic policy?

- We can always go back to good old actor-critic methods
- Getting actor-critic methods to work even without neural networks is hard
- If we want to use on-policy methods, we can't use experience replay buffers
- As usual, simply implementing them with neural networks can be quite unstable
  - And if we want to use experience replay buffers, we need off-policy learning
- Main idea: Train multiple agents simultaneously so data distribution is parallelized and then just use the data generated from different environments in the minibatch

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

Advantage Actor-Critic (A2C)

[Mnih et al., 2016]

- Instead of using the action value function, Q(s,a), for the critic, using only the advantage function, A(s,a) = Q(s,a) - V(s), stabilizes learning
  - The advantage is a relative metric, as we are subtracting the mean from the estimate
  - Is taking an action better or worse than the mean?
  - If A(s,a) > 0, we want to do more of that (thus, gradient goes in that direction)
    If A(s,a) < 0, we want to do less of that (thus, gradient goes against that direction)
- Do we need two value functions, though? Q(s,a) and V(s)?

A(s,a) = Q(s,a) - V(s) *but remember Q(s,a) = r + γV(s')*

Thus, A(s,a) = r + γV(s') - V(s)

A2C is quite close to standard Actor-Critic Methods

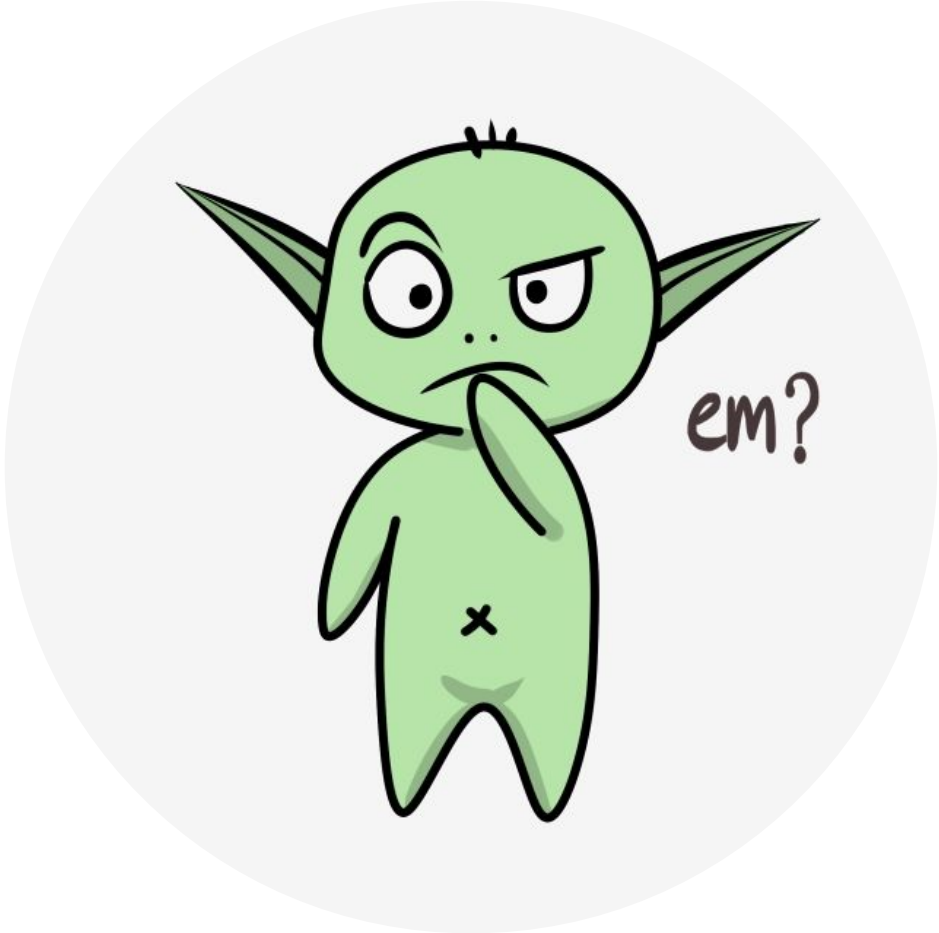
[Mnih et al., 2016]

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

$$\delta_t = R_{t+1} + \gamma V(S_{t+1};\boldsymbol{\theta}) - V(S_t;\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \delta_t \nabla_{\boldsymbol{\theta}} V(S_t;\boldsymbol{\theta})$$

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha \nabla_{\boldsymbol{\phi}} \log \pi(A_t|S_t;\boldsymbol{\phi}) A(S_t, A_t)$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Can we be faster? Can we “cheat”?

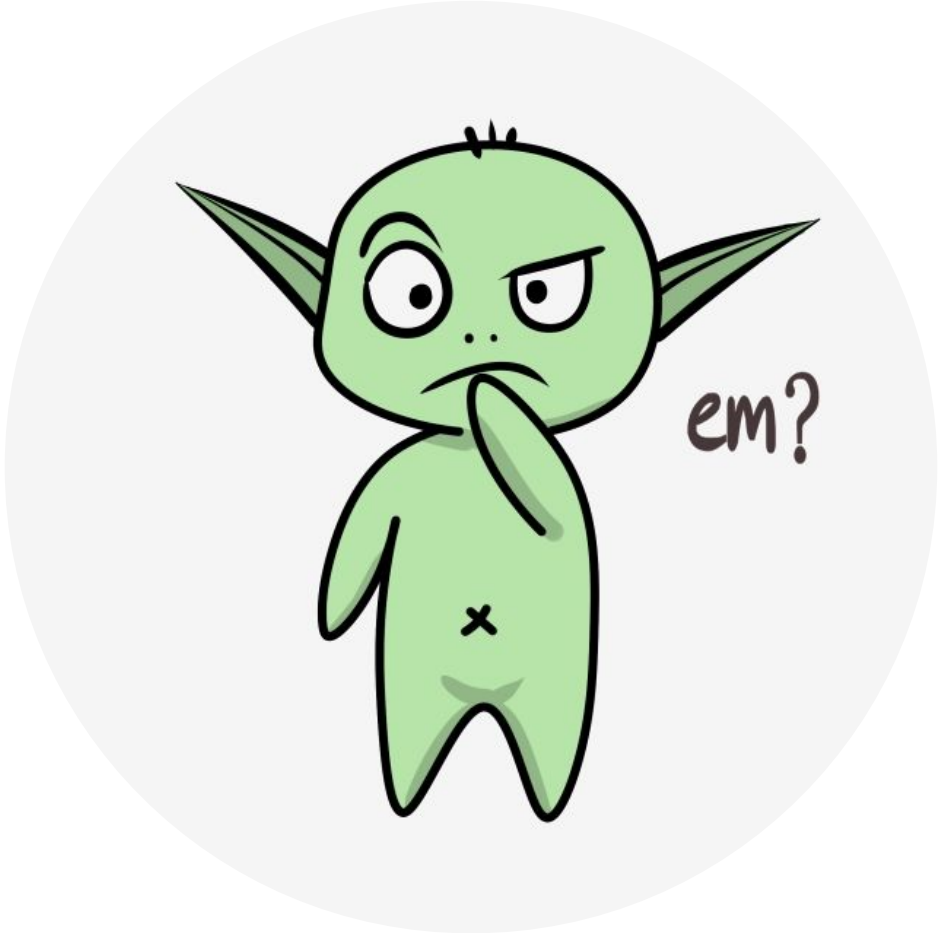
- Can we generate results faster in the problems people have been looking at?
- Can we avoid using GPUs?
- What if we wanted to parallelize learning using distributed computation?
  - Again, notice that this can be seen as a change in the problem formulation
    Maybe it is just accelerating data generation, but it does rely on very strong assumptions
  - “we found that giving each thread a different exploration policy helps improve robustness”

Asynchronous Advantage Actor-Critic [Mnih et al., 2016]

- Key idea: Have many actors generating data in parallel while updating a shared model, the critic
- Implementation-wise, they use multiple CPU threads (no GPU!), removing communication costs as in approaches such as Gorila
- Some consequences of such an approach:
  - Exploration: Actors might end up exploring different parts of the state space
  - Getting rid of the experience replay buffer: The multiple actors already “decorrelate” the data **This makes on-policy learning possible**

Additional details for A3C [Mnih et al., 2016]

- They accumulate gradients over multiple timesteps, reducing the chance different actor learners cancel each other's updates
- Different actors follow different exploration policies
  - ε-greedy with periodically sampled ε from some distribution by each actor
- More algorithmic details
  - n-step return
  - Recurrent agent with an additional 256 LSTM cells after the final hidden layer
  - Entropy regularization w.r.t. policy
  - Shared network parameters between actor and critic (all non-output layers are shared)
  - Share statistics of the gradient across threads for RMSProp

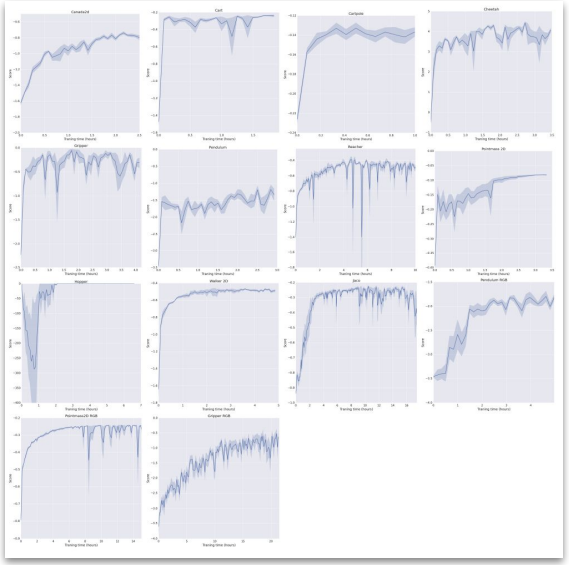


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Empirical evaluation of A3C [Mnih et al., 2016]

- Evaluated in Atari 2600 games, MuJoCo, TORCS, and Labyrinth
  - In Atari 2600 games, the output layer of the policy network was a softmax
    In MuJoCo, the policy network outputs two real numbers, the mean and variance of a multidimensional normal distribution

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%



A3C [Mnih et al., 2016] is faster

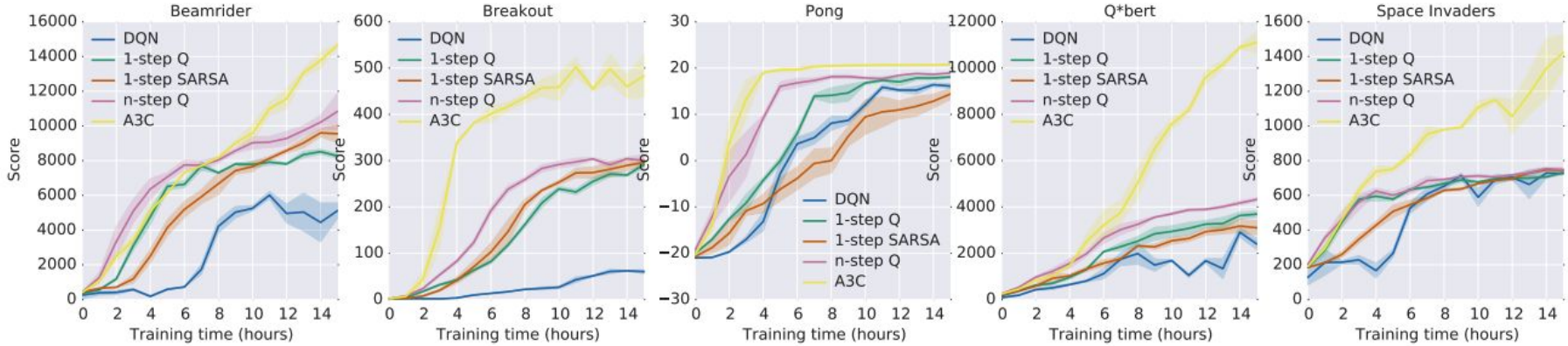


Figure 1. Learning speed comparison for DQN and the new asynchronous algorithms on five Atari 2600 games. DQN was trained on a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores. The plots are averaged over 5 runs. In the case of DQN the runs were for different seeds with fixed hyperparameters. For asynchronous methods we average over the best 5 models from 50 experiments with learning rates sampled from $LogUniform(10^{-4}, 10^{-2})$ and all other hyperparameters fixed.

Modern implementations still use GPUs for neural network computation because training with GPUs is still faster

Some approaches [Mnih et al., 2016] even reach superlinear speedups

- At first, you might expect that the number of steps to achieve the same performance would be the same. Thus, if you have two threads and are generating twice as much data, you'd learn 2x faster. However, it seems that multiple threads can lead to superlinear speedup
  - "We believe this is due to positive effect of multiple threads to reduce the bias in one-step methods."

	Number of threads				
Method	1	2	4	8	16
1-step Q	1.0	**3.0**	**6.3**	**13.3**	**24.1**
1-step SARSA	1.0	**2.8**	**5.9**	**13.1**	**22.1**
n-step Q	1.0	**2.7**	**5.9**	**10.7**	**17.2**
A3C	1.0	2.1	3.7	6.9	12.5

Table 2. The average training speedup for each method and number of threads averaged over seven Atari games. To compute the training speed-up on a single game we measured the time to required reach a fixed reference score using each method and number of threads. The speedup from using n threads on a game was defined as the time required to reach a fixed reference score using one thread divided the time required to reach the reference score using n threads. The table shows the speedups averaged over seven Atari games (Beamrider, Breakout, Enduro, Pong, Q*bert, Seaquest, and Space Invaders).

It is said those methods are also more robust [Mnih et al., 2016]

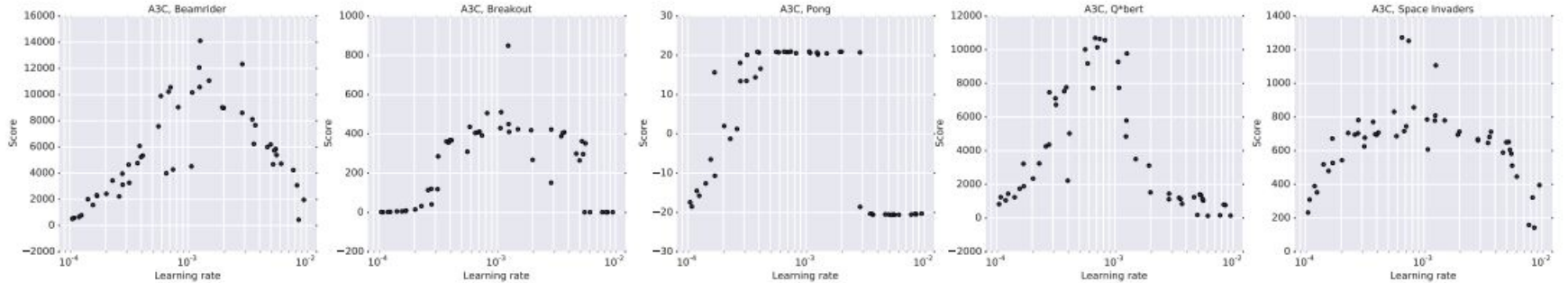
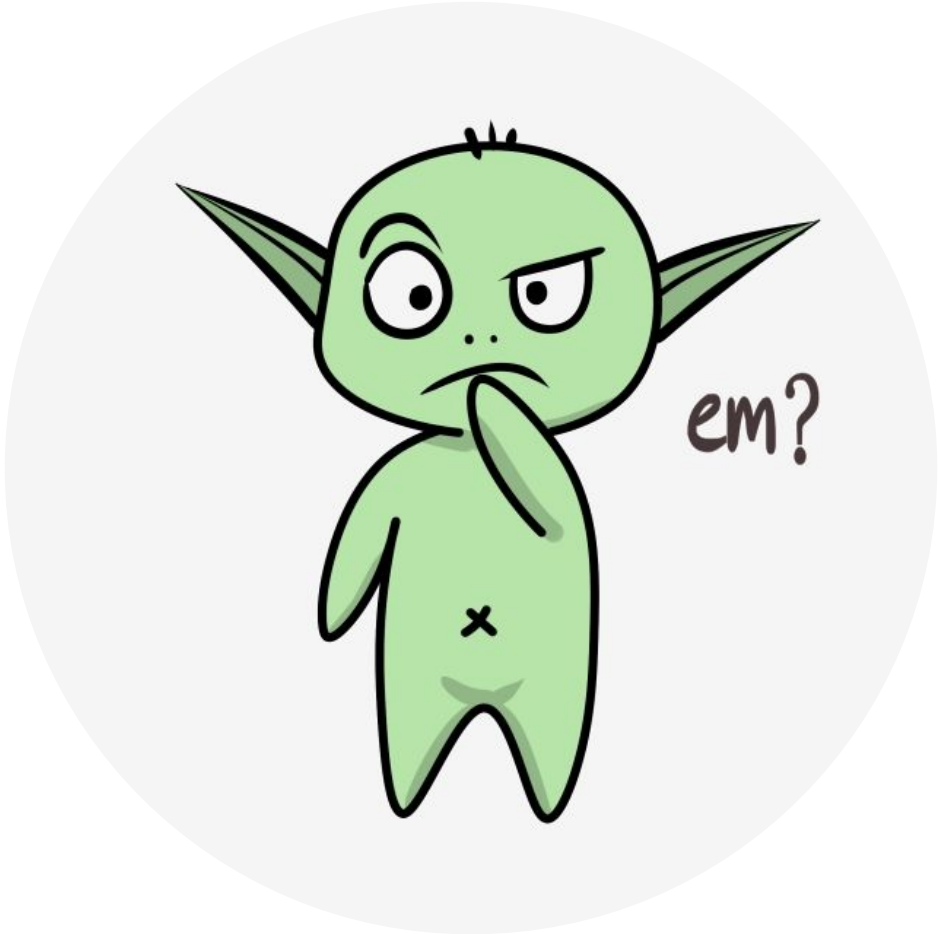


Figure 2. Scatter plots of scores obtained by asynchronous advantage actor-critic on five games (Beamrider, Breakout, Pong, Q*bert, Space Invaders) for 50 different learning rates and random initializations. On each game, there is a wide range of learning rates for which all random initializations acheive good scores. This shows that A3C is quite robust to learning rates and initial random weights.

“Apples-to-apples” comparison in terms of # steps [Mnih et al., 2016]

Game	DQN	Gorila	Double	Dueling	Prioritized	A3C FF, 1 day	A3C FF	A3C LSTM
Alien	570.2	813.5	1033.4	**1486.5**	900.5	182.1	518.4	945.3
Amidar	133.4	189.2	169.1	172.7	218.4	**283.9**	263.9	173.0
Assault	3332.3	1195.8	6060.8	3994.8	7748.5	3746.1	5474.9	**14497.9**
Asterix	124.5	3324.7	16837.0	15840.0	**31907.5**	6723.0	22140.5	17244.5
Asteroids	697.1	933.6	1193.2	2035.4	1654.0	3009.4	4474.5	**5093.1**
Atlantis	76108.0	629166.5	319688.0	445360.0	593642.0	772392.0	**911091.0**	875822.0
Bank Heist	176.3	399.4	886.0	**1129.3**	816.8	946.0	970.1	932.8
Battle Zone	17560.0	19938.0	24740.0	**31320.0**	29100.0	11340.0	12950.0	20760.0
Beam Rider	8672.4	3822.1	17417.2	14591.3	**26172.7**	13235.9	22707.9	24622.2
Berzerk			1011.1	910.6	1165.6	**1433.4**	817.9	862.2
Bowling	41.2	54.0	**69.6**	65.7	65.8	36.2	35.1	41.8
Boxing	25.8	74.2	73.5	**77.3**	68.6	33.7	59.8	37.3
Breakout	303.9	313.0	368.9	411.6	371.6	551.6	681.9	**766.8**
Centipede	3773.1	**6296.9**	3853.5	4881.0	3421.9	3306.5	3755.8	1997.0
Chopper Comman	3046.0	3191.8	3495.0	3784.0	6604.0	4669.0	7021.0	**10150.0**
Crazy Climber	50992.0	65451.0	113782.0	124566.0	131086.0	101624.0	112646.0	**138518.0**
Defender			27510.0	33996.0	21093.5	36242.5	56533.0	**233021.5**
Demon Attack	12835.2	14880.1	69803.4	56322.8	73185.8	84997.5	113308.4	**115201.9**
Double Dunk	-21.6	-11.3	-0.3	-0.8	**2.7**	0.1	-0.1	0.1
Enduro	475.6	71.0	1216.6	**2077.4**	1884.4	-82.2	-82.5	-82.5
Fishing Derby	-2.3	4.6	3.2	-4.1	9.2	13.6	18.8	**22.6**
Freeway	25.8	10.2	**28.8**	0.2	27.9	0.1	0.1	0.1
Frostbite	157.4	426.6	1448.1	2332.4	**2930.2**	180.1	190.5	197.6
Gopher	2731.8	4373.0	15253.0	20051.4	**57783.8**	8442.8	10022.8	17106.8
Gravitar	216.5	**538.4**	200.5	297.0	218.0	269.5	303.5	320.0
H.E.R.O.	12952.5	8963.4	14892.5	15207.9	20506.4	28765.8	**32464.1**	28889.5
Ice Hockey	-3.8	-1.7	-2.5	-1.3	**-1.0**	-4.7	-2.8	-1.7
James Bond	348.5	444.0	573.0	835.5	**3511.5**	351.5	541.0	613.0
Kangaroo	2696.0	1431.0	**11204.0**	10334.0	10241.0	106.0	94.0	125.0
Krull	3864.0	6363.1	6796.1	8051.6	7406.5	**8066.6**	5560.0	5911.4
Kung-Fu Master	11875.0	20620.0	30207.0	24288.0	31244.0	3046.0	28819.0	**40835.0**
Montezuma's Revenge	50.0	**84.0**	42.0	22.0	13.0	53.0	67.0	41.0
Ms. Pacman	763.5	1263.0	1241.3	**2250.6**	1824.6	594.4	653.7	850.7
Name This Game	5439.9	9238.5	8960.3	11185.1	11836.1	5614.0	10476.1	**12093.7**
Phoenix			12366.5	20410.5	27430.1	28181.8	52894.1	**74786.7**
Pit Fall			-186.7	-46.9	**-14.8**	-123.0	-78.5	-135.7
Pong	16.2	16.7	**19.1**	18.8	18.9	11.4	5.6	10.7
Private Eye	298.2	**2598.6**	-575.5	292.6	179.0	194.4	206.9	421.1
Q*Bert	4589.8	7089.8	11020.8	14175.8	11277.0	13752.3	15148.8	**21307.5**
River Raid	4065.3	5310.3	10838.4	16569.4	**18184.4**	10001.2	12201.8	6591.9
Road Runner	9264.0	43079.8	43156.0	58549.0	56990.0	31769.0	34216.0	**73949.0**
Robotank	58.5	61.8	59.1	**62.0**	55.4	2.3	32.8	2.6
Seaquest	2793.9	10145.9	14498.0	37361.6	**39096.7**	2300.2	2355.4	1326.1
Skiing			-11490.4	-11928.0	**-10852.8**	-13700.0	-10911.1	-14863.8
Solaris			810.0	1768.4	**2238.2**	1884.8	1956.0	1936.4
Space Invaders	1449.7	1183.3	2628.7	5993.1	9063.0	2214.7	15730.5	**23846.0**
Star Gunner	34081.0	14919.2	58365.0	90804.0	51959.0	64393.0	138218.0	**164766.0**
Surround			1.9	**4.0**	-0.9	-9.6	-9.7	-8.3
Tennis	-2.3	-0.7	-7.8	**4.4**	-2.0	-10.2	-6.3	-6.4
Time Pilot	5640.0	8267.8	6608.0	6601.0	7448.0	5825.0	12679.0	**27202.0**
Tutankham	32.4	118.5	92.2	48.0	33.6	26.1	**156.3**	144.2
Up and Down	3311.3	8747.7	19086.9	24759.2	29443.7	54525.4	74705.7	**105728.7**
Venture	54.0	**523.4**	21.0	200.0	244.0	19.0	23.0	25.0
Video Pinball	20228.1	112093.4	367823.7	110976.2	374886.9	185852.6	331628.1	**470310.5**
Wizard of Wor	246.0	10431.0	6201.0	7054.0	7451.0	5278.0	17244.0	**18082.0**
Yars Revenge			6270.6	**25976.5**	5965.1	7270.8	7157.5	5615.5
Zaxxon	831.0	6159.4	8593.0	10164.0	9501.0	2659.0	**24622.0**	23519.0

- Number of “wins”:
  - DQN: 0
  - Gorila: 5
  - DDQN: 4
  - Dueling: 9
  - Prioritized: 12
  - A3C FF: 4
  - A3C LSTM: 19
- It seems you can train faster, but you can’t necessarily reach better performance (but maybe you can better use an LSTM)



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Next class

- What I plan to do:
  - Wrap up the discussion about policy gradient methods, covering TRPO, PPO (and GAE) & SAC
- What I recommend you to do for next class:
  - Assignments, as usual :-)
  - Read
    - J. Schulman et al.: Trust Region Policy Optimization. ICML, 2015.
    - J. Schulman et al.: Proximal Policy Optimization Algorithms. CoRR abs/1707.06347, 2017.
    - J. Schulman et al.: High-Dimensional Continuous Control Using Generalized Advantage Estimation. ICLR, 2016
    - T. Haarnoja et al.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. ICML, 2018.