Fraunhofer
IIS

Fraunhofer-Institut für Integrierte Schaltungen IIS

Reinforcement Learning

Lecture 7: Policy-based RL 2

Christopher Mutschler

Policy-based Reinforcement Learning 1

Recap

- But how do we maximize this?
  → **Gradient Ascent!** *Suppose* we know how to calculate the gradient w.r.t. the parameters:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)$$

- Then we can update our parameters θ with a learning rate α in the direction of the gradient:

$$\theta \leftarrow \theta + \alpha \underline{\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)}$$

Policy Gradient

often in literature referred to as $\nabla_\theta J(\pi_\theta)$

Fraunhofer IIS

Policy-based Reinforcement Learning 1

Recap

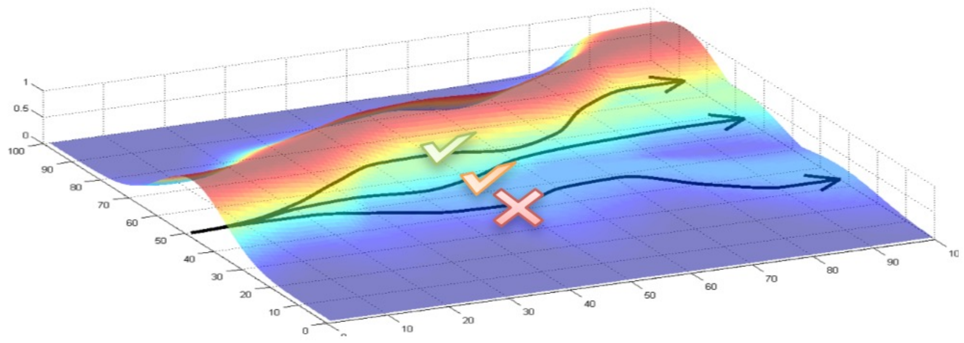
- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- But what is the intuition behind this gradient?

- The gradient tries to
  - Increase probability of paths with positive G
  - Decrease probability of paths with negative G

  In practice:
  - Increase probability of paths with small positive G a bit
  - Increase probability of paths with large positive G much more



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

Fraunhofer IIS

Policy-based Reinforcement Learning 1
Recap

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- **Reduce variance:** as this is an expectation, we can estimate it with a sample mean using Monte-Carlo sampling of L trajectories:

each action in the episode in influenced by the reward of the whole episode???
→ *reward-to-go* policy gradient

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)$$
$$\approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})$$

Policy-based Reinforcement Learning1

Recap

- Monte-Carle Policy Gradient Control
  - Update parameters by stochastic gradient ascent
  - Using policy gradient theorem
  - Using return-to-go $Q^{\pi_\theta}(s_t, a_t)$ as an unbiased sample of G:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)\, \gamma G_t$$

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

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy-based RL
Agenda

- Policy-based RL 1
  - Intro to Policy-based RL
  - Policy Gradients

- Policy-based RL 2
  - **Variance & Baselines**
  - Actor-Critics
  - Trust-Region Policy Optimization (TRPO)
  - Proximal Policy Optimization (PPO)
  - Deep Deterministic Policy Gradient (DDPG)

Policy-based Reinforcement Learning

Policy Gradients: where to hide the variance?

DIGRESSION

- Expected Grad-Log-Prob Lemma*:
  - P_θ is a parameterized probability distribution over a random variable x, then:

$$\mathbb{E}_{x \sim P_\theta}[\nabla_\theta \log P_\theta(x)] = 0$$

- From this follows that for any function b that *only depends on states*:

$$\mathbb{E}_{a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = 0,$$

because:

$$= \mathbb{E}\left[\sum_a \pi(a|s_t) b \nabla_\theta \log \pi(a|s_t)\right]$$
$$= \mathbb{E}\left[b \nabla_\theta \sum_a \pi(a|s_t)\right]$$
$$= \mathbb{E}[b \nabla_\theta 1]$$
$$= 0$$

inverse
score-function trick

prob-distribution and
we sum over all actions

* *You can find one version of the proof here: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#id1*

Policy-based Reinforcement Learning
Policy Gradients: where to hide the variance?

DIGRESSION

- This allows us to add or subtract any of such terms (i.e., **baseline functions**) to our policy gradient without changing its expectation:

$$= \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'}) = Q^{\pi}(s_t, a_t)$$

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau) = \mathbb{E}_{\tau \sim \pi_\theta} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \left(G_t - b(s_t) \right) \right)$$

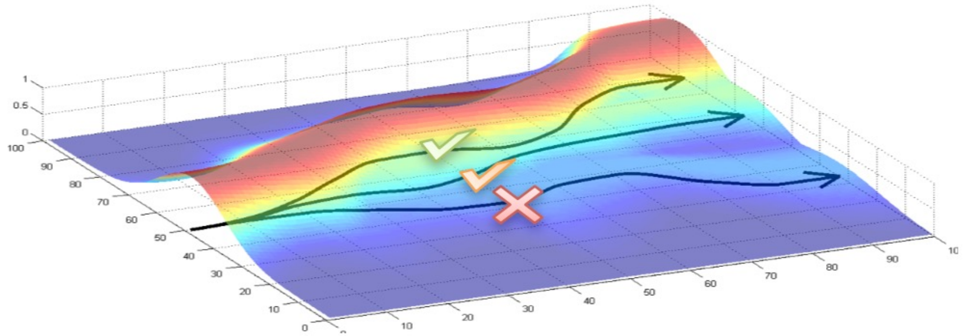
* You can find one version of the proof here: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#id1

Policy-based Reinforcement Learning

Advantage Functions: Intuition

- But what is the intuition behind this gradient?

- The gradient tries to
  - Increase probability of paths with positive G
  - Decrease probability of paths with negative G



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

- You mostly keep on increasing everything (but some more than others)
  - And this requires a lot of rollouts to average the effect out!
  - Ideal: in-/decrease probs of paths that are better/worse than the average!

Policy-based Reinforcement Learning

Advantage Functions: Introduce a Baseline

But how does this help?

- We can introduce a baseline to our targets!
- We can define *advantage functions* that reduce variance
- Intuition: if an agent sees what it expected, it feels *neutral* about it
- *Hint: we already did this when we introduced the reward-to-go variant!*

- For instance, we can use $b(s_t) = v^{\pi}(s_t)$ to reduce variance in the sample estimate of the policy gradient:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

→ faster and more stable policy learning!

Fraunhofer IIS

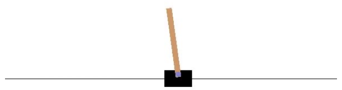
Policy-based Reinforcement Learning

Advantage Functions: Intuition

Question: what helps our agent learn faster: information about

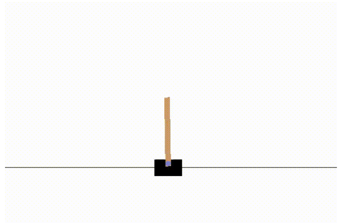
1. *invariant actions on good states*, or
2. information about *good actions in bad/challenging states*?

<u>Iteration 100</u>



$Q(s2, \text{right}) = 50$
$V(s2) = 10$
$A(s2, \text{right}) = 40$

<u>Iteration 1000</u>



$Q(s1, \text{do nothing}) = 100$
$V(s1) = 100$
$A(s1, \text{do nothing}) = 0$

Policy-based Reinforcement Learning

REINFORCE w/ Baselines

REINFORCE with Baseline (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 $\hat{v}(s,\mathbf{w})$
Algorithm 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):
    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)
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$ ← *advantage*
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$ ← policy gradient update

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy-based Reinforcement Learning

REINFORCE w/ Baselines

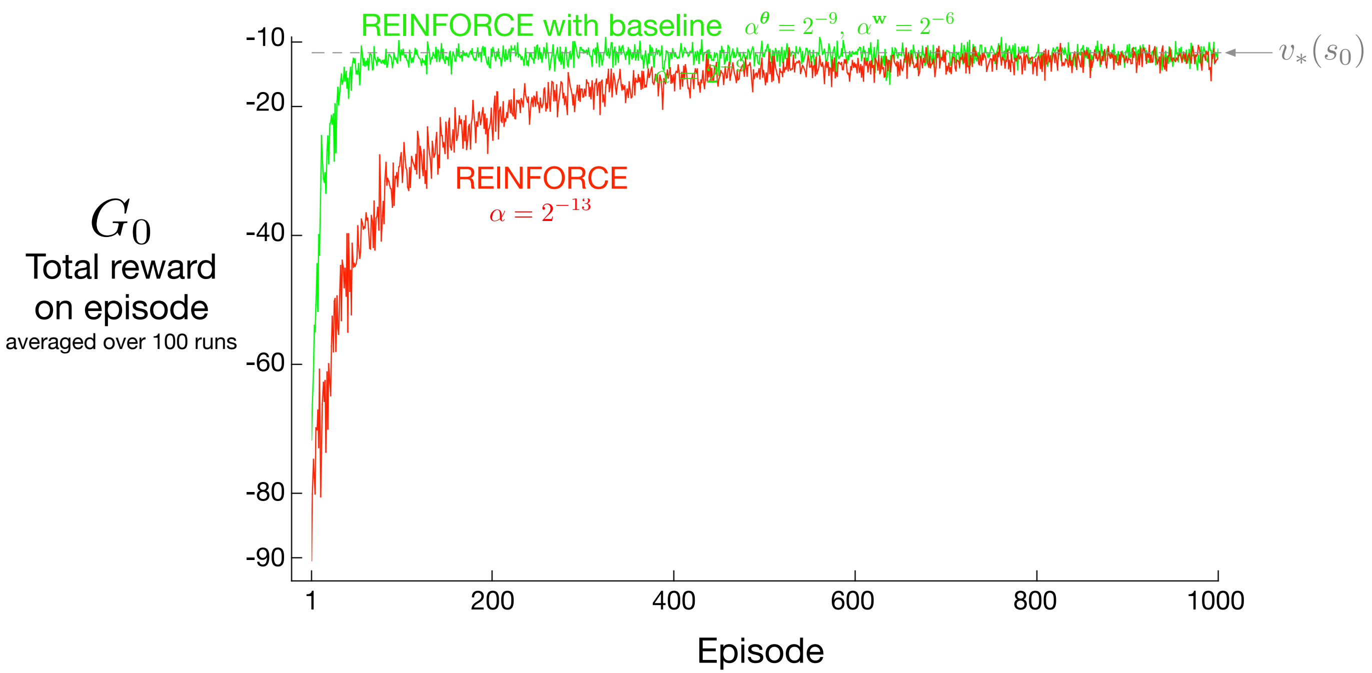


Figure 13.2: Adding a baseline to REINFORCE can make it learn much faster, as illustrated here on the short-corridor gridworld (Example 13.1). The step size used here for plain REINFORCE is that at which it performs best (to the nearest power of two; see Figure 13.1).

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Fraunhofer IIS

Policy-based Reinforcement Learning

Vanilla Policy Gradient Algorithm

Problem:
- Learning rate is a heuristic that cannot be easily tuned
- If we take large steps (especially in the beginning of training) we might never recover

Pseudocode

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \boxed{\alpha_k \hat{g}_k},$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

Estimate the Advantages

Calculate the gradient and take a gradient step

Update the Critic (used for the Advantage estimation)

Policy-based RL

Agenda

- Policy-based RL 1
  - Intro to Policy-based RL
  - Policy Gradients

- Policy-based RL 2
  - Variance & Baselines
  - **Actor-Critics**
  - Trust-Region Policy Optimization (TRPO)
  - Proximal Policy Optimization (PPO)
  - Deep Deterministic Policy Gradient (DDPG)

Actor-Critics

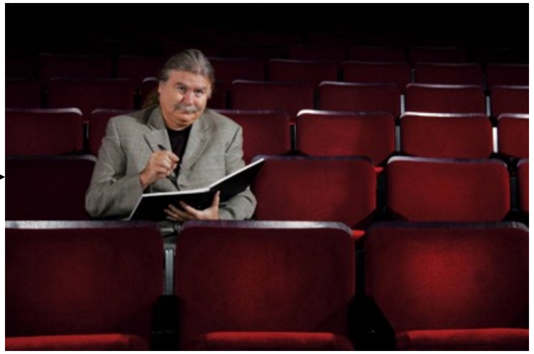
Policy Gradients: Variance (revisited)

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L} \sum_\tau \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \underbrace{\sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})}_{= \, Q^\pi(s_t, a_t)} - b(s_t)$$

- Monte-Carlo policy gradient is sampled and has high variance
- Idea: we can use a critic that estimates the Q

actor

critic



Actor-Critics

Policy Gradients: Variance (revisited)

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \underbrace{\sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})}_{= \, Q^\pi(s_t, a_t)} - b(s_t)$$

- Monte-Carlo policy gradient is sampled and has high variance
- Idea: we can use a critic that estimates the Q

→ In practice, (as we already know) $Q^\pi(s_t, a_t)$ cannot be "computed"

→ we instead need to approximate it with a neural network with parameters ϕ and standard SGD over k epochs minimizing the MSE:

$$\phi_k = \arg\min_{\phi} \mathbb{E}_{s_t; a_t, \hat{R}_t \sim \pi_k} \left[\left(Q_\phi(s_t, a_t) - \hat{R}_t \right)^2 \right]$$

Actor-Critics

Actor-Critic

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L} \sum_\tau \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \underbrace{\sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})}_{= \; Q^\pi(s_t, a_t)} - b(s_t)$$

- Monte-Carlo policy gradient is sampled and has high variance
- Idea: we can use a critic (e.g., a NN as in DQN) to estimate the Q and learn two sets of parameters separately
  - Actor: update θ by policy gradient
  - Critic: Update parameters ϕ of $\boxed{v_\phi^\pi,}$ e.g., by n-step TD ?

- We call such algorithms *actor-critic algorithms*

Actor-Critics

Actor-Critic

- Typically, we estimate $v^{\pi}(s_t;\phi)$ explicitly, and then sample

$$q^{\pi}(s_t, a_t) \approx G_t^{(n)}$$

- For instance: $\hat{G}_t^{(1)} = R_t + \gamma v^{\pi}(s_{t+1};\phi)$

- Then we arrive at:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} \hat{G}(\tau) = \frac{1}{L} \sum_{\tau} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\,(\hat{G}_t - b(s_t))$$

→ We use an *approximate* gradient in the direction suggested by the critic

Fraunhofer IIS

Actor-Critics

Actor-Critic: Advantage Functions

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \boxed{(\hat{G}_t - b(s_t))}$$

$$:= A^\pi(s_t, a_t)$$

Calculating the advantage function is (embarrassingly) straight forward:

- As before, we can simply use the TD error:

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r + \gamma \cdot v^\pi(s_{t+1}) - v^\pi(s_t) \end{aligned}$$

Actor-Critics

Actor-Critic: Advantage Functions

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\,\boxed{(\hat{G}_t - b(s_t))}$$

$$:= A^\pi(s_t, a_t)$$

Calculating the advantage function is (embarrassingly) straight forward:

- We can also use **Generalized Advantage Estimation (GAE)** (multi-step TD-error like TD with n-step returns)

$$\hat{A}_t^{(1)} \coloneqq \delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1})$$

$$\hat{A}_t^{(2)} \coloneqq \delta_t^V + \gamma \delta_{t+1}^V = -V(s_t) + r_t + \gamma V(s_{t+1}) + \gamma^2 V(s_{t+2})$$

$$\hat{A}_t^{(3)} \coloneqq \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V = -V(s_t) + r_t + \gamma V(s_{t+1}) + \gamma^2 V(s_{t+2}) + \gamma^3 V(s_{t+3})$$

$$\dots$$

$$\hat{A}_t^{(k)} \coloneqq \textstyle\sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$$

$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l + r_{t+l}$$

Actor-Critics

Actor-Critic: Advantage Functions

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \, \underbrace{\boxed{(\hat{G}_t - b(s_t))}}_{:= A^\pi(s_t, a_t)}$$

Calculating the advantage function is (embarrassingly) straight forward:

- Full returns: high variance
- One-step TD: high bias
- n-step TD: somewhere in the middle

But still: approximating the policy gradient introduces bias

- It is important to use on-policy targets (i.e., can be corrected using importance sampling)
- Alternative idea: bootstrap (with $\lambda = 0$) whenever policies differ

Actor-Critics

Continuous Actions: Gaussian Policy

- Because we directly update the policy parameters of the policy, we can easily deal with continuous action spaces
  - Most algorithms discussed can be used for both discrete and continuous actions

- In continuous action spaces, a Gaussian policy is common, e.g., mean is some function of state $\mu(s)$

- For simplicity, lets consider fixed variance of σ^2 (which can be parameterized as well, instead)

- Policy is Gaussian: $a \sim \mathcal{N}(\mu(s), \sigma^2)$

- The gradient of the log of the policy is then

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

- This can be used, for instance, in REINFORCE or advantage actor-critic

Fraunhofer IIS

Actor-Critics

Continuous Actions: Exploration-Exploitation

- Exploration vs. Exploitation of Policy Gradient methods:
  - PG trains a stochastic policy in an on-policy way
  - Actions are sampled from the environment according to the latest version of its stochastic policy
  - Randomness in selecting actions
    - (initially) depends on the initialization
    - Becomes less over the course of training
    - …as the update rule encourages to exploit rewards that the policy already has found

  - *Policy-gradients only consider improvement under current data*
    → **easy to get stuck in local optima**

Actor-Critics

One more thing: Exploration-Exploitation

- Could we use ϵ-greedy? – Yes! but it is not ideal…
  - Wildly different actions cause breakage
  - Exploration is mostly uninformed about current best guess

- Alternative idea: make sure that the entropy of the policy is not too low:

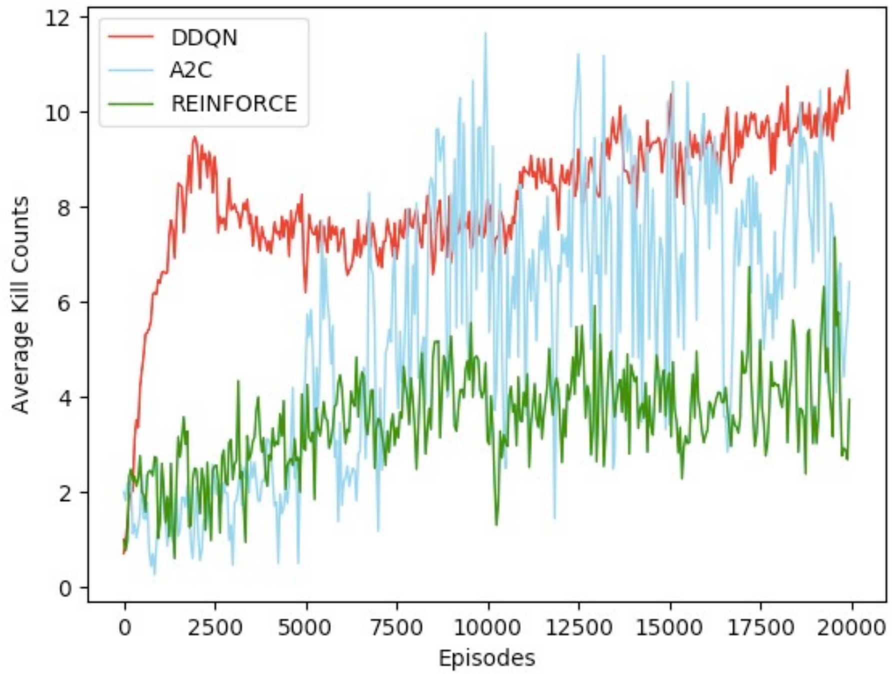
$$-\sum_s \mu(s) \sum_a \pi(a|s) \log \pi (a|s) = -\mathbb{E}[\log \pi(a_t|s_t)]$$

  → Add a regularization term that pushes entropy up slightly each step
  - Encourages exploration and does not pick fully randomly
    - May increase variance in Gaussian policies
    - Makes softmax slightly more uniform
  - Similar to (spoiler!) KL-regularization (in TRPO)
  - Works well in practice

Fraunhofer IIS

Actor-Critics
Conclusion

- It is substantially different from DQNs
  - no replay buffer, no stored experiences
  - learn directly, on-policy



https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html

- Once a batch has been used → discard experience
  - less sample efficient

- Learning off-policy (which would allow to reuse experience) is not (or only with certain tricks) possible

Fraunhofer IIS

Actor-Critics

Conclusion

- The policy gradient has many forms:

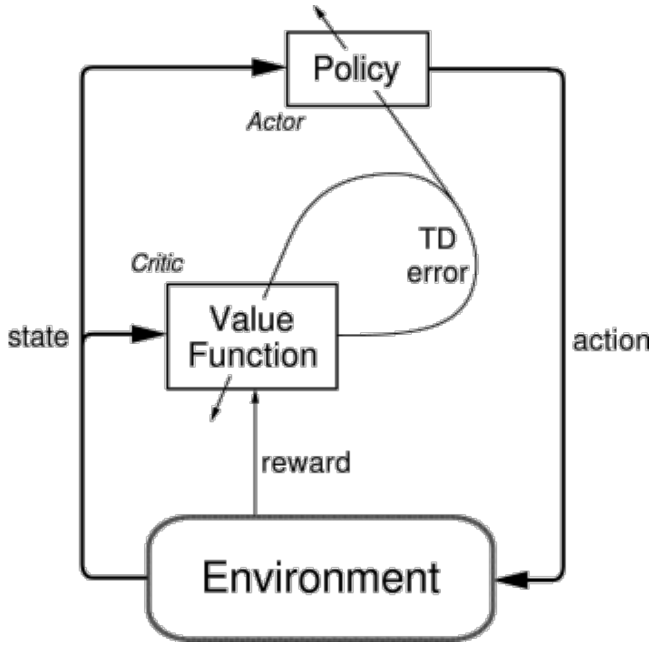
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t]$$ REINFORCE

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) (G_t - b(s_t))]$$ REINFORCE w/ baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \hat{A}_t^{(\infty)}\right]$$ advantage actor-critic

$$\nabla_\theta J(\theta) = \nabla_\theta Q_t(s, \pi_\theta(s))$$ deterministic policy gradient (see DDPG)

- Each leads to a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g., MC or TD) to estimate $Q^\pi(s,a)$ or $V^\pi(s)$



Sutton et a.: Reinforcement learning: An introduction. 2018.

Fraunhofer IIS

Actor-Critics

Lessons Learned

"Always try to solve a problem the direct way
– but be careful with the high variance."

https://www.youtube.com/watch?v=ek3hgVQ1RqM

Policy-based RL
Agenda

- Policy-based RL 1
  - Intro to Policy-based RL
  - Policy Gradients

- Policy-based RL 2
  - Variance & Baselines
  - Actor-Critics
  - **Trust-Region Policy Optimization (TRPO)**
  - Proximal Policy Optimization (PPO)
  - Deep Deterministic Policy Gradient (DDPG)

Trust-Region Policy Optimization

Policy Gradients so far

- We wanted to do the following:

$$\max_{\theta} J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

- Then we defined the policy gradient:

$$g = \nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\,\hat{A}_t$$

- If $\hat{A}_t > 0$ (< 0):
  - Gradient becomes positive (negative)
  - The probability of taking a_t in s_t is increased (decreased)

- We update our policy parameters by

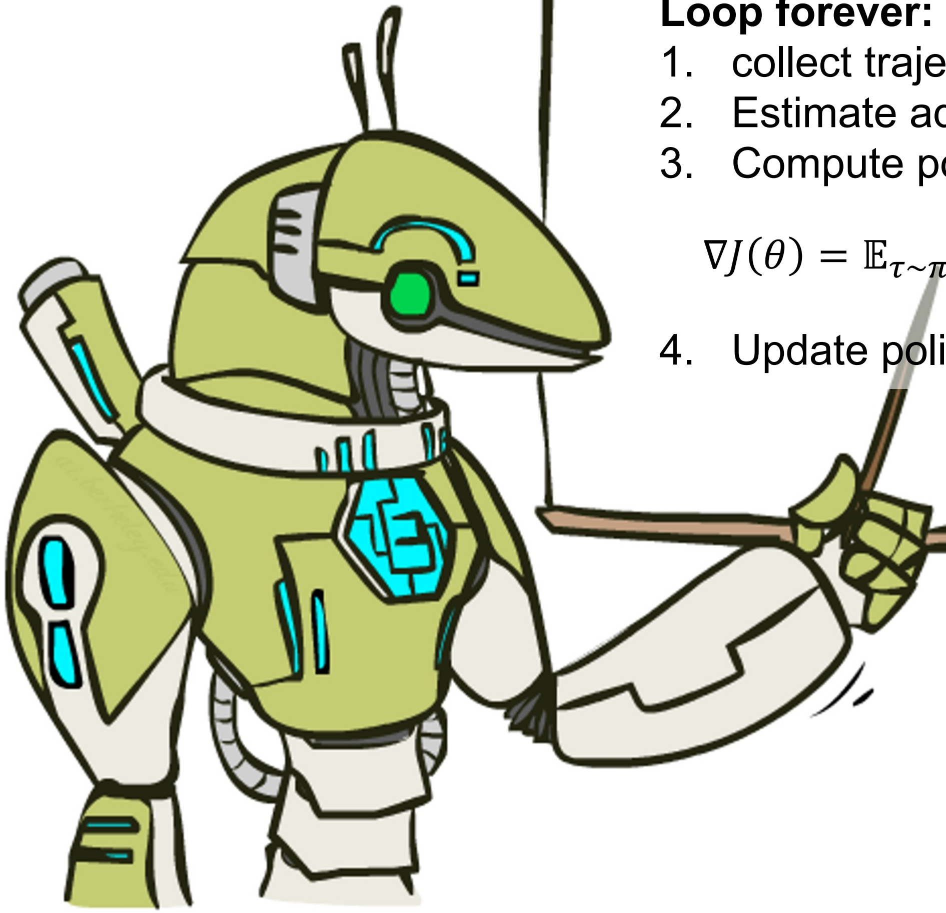
$$\theta_{k+1} = \theta_k + \alpha \cdot g$$

Take a gradient step in updating the policy (gradient ascent)

Fraunhofer IIS

Trust-Region Policy Optimization

Policy Gradients so far



Loop forever:

1. collect trajectories via policy π_θ
2. Estimate advantage function $A^{\pi_\theta}(a_t|s_t)$
3. Compute policy gradient:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, A^{\pi_\theta}(a_t|s_t)\right)$$

4. Update policy parameters $\theta_{new} \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

http://ai.berkeley.edu/lecture_slides.html

Fraunhofer IIS

Trust-Region Policy Optimization

Policy Gradients so far

Problem

- Run gradient descent/ascent on one batch of collected experience
- Note: the advantage function (which is a noisy estimate) may not be accurate
  - Too large steps may lead to a disaster (even *if* the gradient is *correct*)
  - Too small steps are also bad
- Definition and scheduling of learning rates in RL is tricky as the underlying data distribution changes with updates to the policy
- Mathematical formulization:
  - First-order derivatives approximate the (parameter) surface to be flat
  - But if the surface exhibits high curvature it gets dangerous
  - Projection: small changes in parameter space might lead to large changes in policy space!
- **Parameters θ get updated to areas too far out of the range from where previous data was collected** ***(note: a bad policy leads to bad data – unlike in value-based RL!)***

Images taken from https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9 and http://www.taiwanoffthebeatentrack.com/2012/08/23/mount-hua-华山-the-most-dangerous-hike-in-the-world/

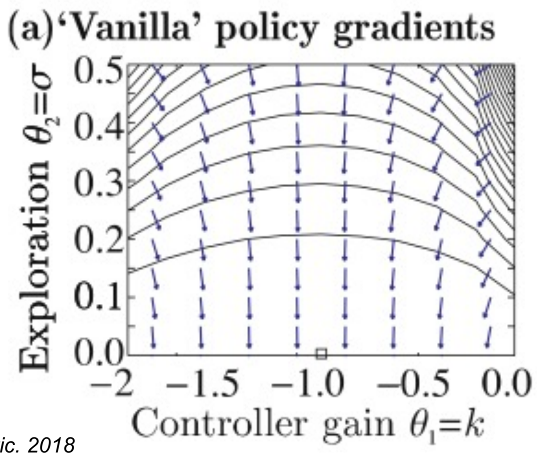
Fraunhofer IIS

Trust-Region Policy Optimization

Primer: Natural Policy Gradient

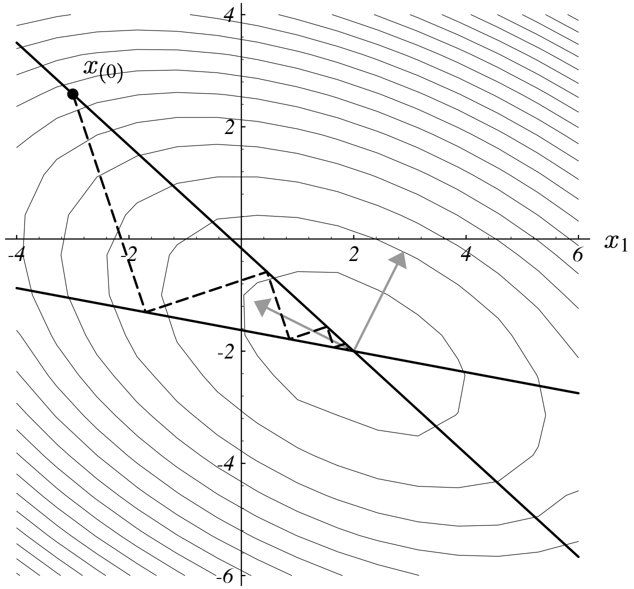
There is something wrong with the policy gradient. Example:

- $r(s_t, a_t) = -s_t^2 - a_t^2$
- $\log \pi_\theta(a_t|s_t) = -\frac{1}{2\sigma^2}(ks_t - a_t)^2 + c$, with $\theta = (k, \sigma)$



Peters et al.: Natural Actor-Critic. 2018

poor conditioning



Shewchuk: An Introduction to the Conjugate Gradient Method without the Agonizing Pain. Edition 1 ¼.

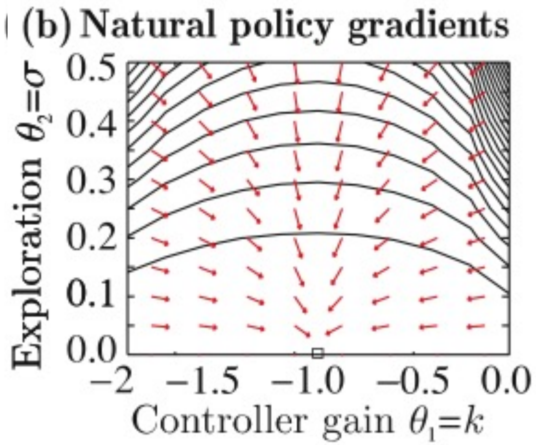
Trust-Region Policy Optimization

Primer: Natural Policy Gradient

- Given:
  - $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ and $\pi_\theta(a_t|s_t)$
- Some parameters change probabilities a lot more than others!

  → a single value for α is not sufficient
- What we essentially do (optimization perspective on 1st order gradient descent):
  - $\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta)$, subject to $\|\theta' - \theta\|^2 \leq \epsilon$
  - $\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta)$, subject to $\boxed{D(\pi_{\theta'}, \pi_\theta) \leq \epsilon}$

“trust region”



But how to rescale???

→ For instance, KL-divergence

Better with 2nd order information

→ $\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta)$, s.t. $\|\theta' - \theta\|_{\mathbf{F}}^2 \leq \epsilon$

→ $\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$

Trust-Region Policy Optimization (TRPO)

<u>“Simple” Idea</u>

Regularize updates to the policy parameters,
such that the policy does not change too much.

Optimization in Machine Learning

- It is common to formulate ML problems as optimization problems:
  - Minimize the squared error
  - Minimize the cross entropy
  - Maximize the log-likelihood
  - Maximize the discounted sum of rewards
  - Minimize the sum of control cost

Trust-Region Policy Optimization

only for reference

Primer: Trust-Region Methods

Optimization in Machine Learning: two classes

1. Line Search, e.g., gradient descent
   - find a (some) direction of improvement
   - (cleverly) select a step length

2. Trust-Region Methods
   - select a trust region (analog to max step length)
   - find a point of improvement in that region

Trust-Region Policy Optimization

Primer: Trust-Region Methods

- **Idea:**
  - Approximate the real objective f with something simpler, i.e., $\tilde{f}$
  - Solve $\tilde{x}^* = \arg\min_x \tilde{f}(x)$
- **Problem:**
  - The optimum $\tilde{x}^*$ might be in a region where $\tilde{f}$ poorly approximates f
  - $\tilde{x}^*$ might be far from optimal
- **Solution:**
  - Restrict the search to a region tr where we trust $\tilde{f}$ to approximate f well
  - Solve $\tilde{x}^* = \arg\min_{x \in tr} \tilde{f}(x)$

Trust-Region Policy Optimization

only for reference

Primer: Trust-Region Methods

2nd order Taylor approximation

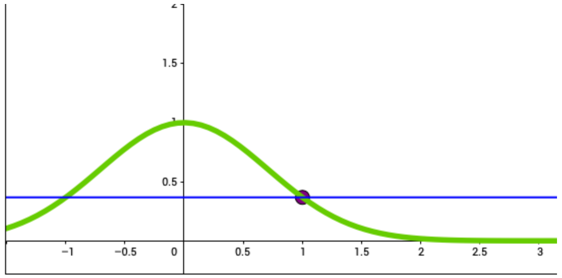
- Example: chose $\tilde{f}$ to be quadratic approximation of f:

$$f(x) \approx \tilde{f}(x) = f(c) + \nabla f(c)^T(x-c) + \frac{1}{2!}(x-c)^T H(c)(x-c),$$

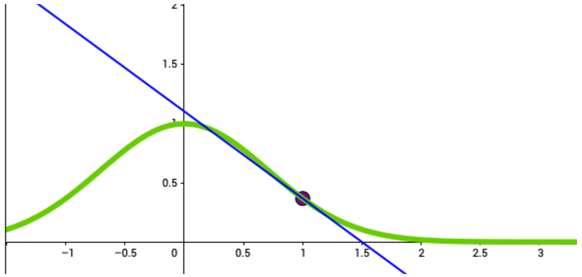
where ∇f is the gradient and H is the Hessian

→ $p_n(x) = \sum_{j=0}^{n} \frac{1}{j!} f^{(j)}(a)(x-a)^j$

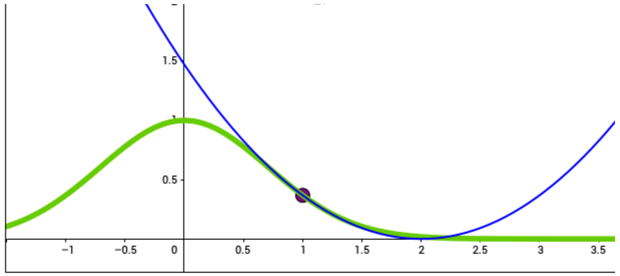
Example: nth order Taylor approximation of $f(x) = e^{-x^2}, c = 1$



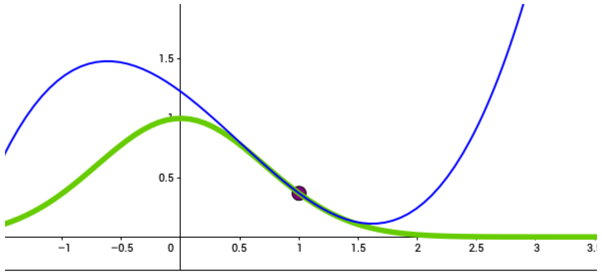
$p_0(x) = 0.37$



$p_1(x) = p_0(x) - 0.74 \cdot (x-1)$



$p_2(x) = p_1(x) + 0.74 \cdot \frac{(x-1)^2}{2!}$



$p_3(x) = p_2(x) + 1.47 \cdot \frac{(x-1)^3}{3!}$

https://mathinsight.org/applet/taylor_polynomial

Fraunhofer IIS

2^{nd} order Taylor approximation

- Example: chose $\tilde{f}$ to be quadratic approximation of f:

$$f(x) \approx \tilde{f}(x) = f(c) + \nabla f(c)^T (x - c) + \frac{1}{2!}(x - c)^T H(c)(x - c),$$

  where ∇f is the gradient and H is the Hessian

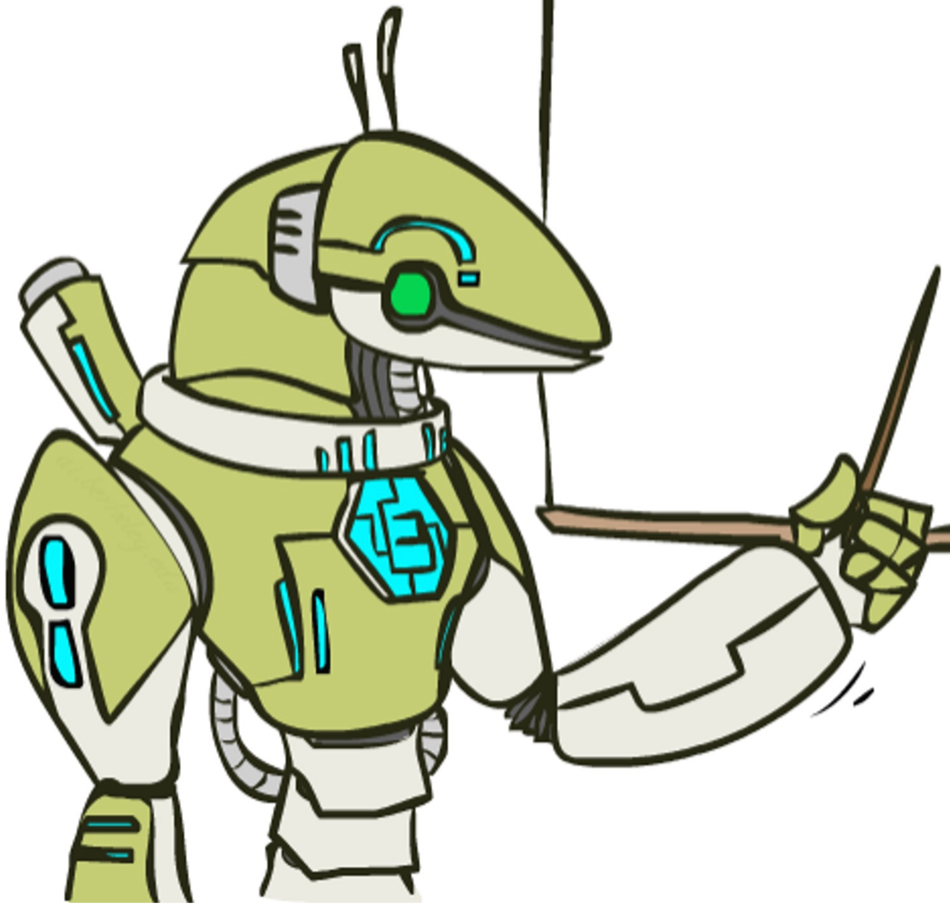
- The trust region is often chosen to be a hypersphere:

$$\|x - c\|_2 \leq \delta$$

Trust-Region Policy Optimization

Primer: Trust-Region Methods

only for reference



Initialize $\delta, x_0^*, n = 0$

Loop forever:

1. $n \leftarrow n + 1$
2. Solve $\mathrm{x}_n^* = \arg\min_x \tilde{f}(x)$
   subject to $\|x - x_{n-1}^*\|_2 \leq \delta$
3. If $\tilde{f}(x) \approx f(x_n^*)$: increase δ,
   else: decrease δ

2nd order Taylor approximation

- $\tilde{f}$ often chosen to be quadratic approximation of f:

$$\min_{x} f(c) + \nabla f(c)^T(x-c) + \frac{1}{2!}(x-c)^T H(c)(x-c),$$

subject to $\|x-c\|_2 \leq \delta$.

- When H is positive semi-definite
  - Convex optimization
  - Simple and globally optimal solution
  - e.g.: conjugate gradient

- When H is not positive semi-definite
  - Non-convex optimization
  - Simple heuristics that guarantee improvement

Fraunhofer IIS

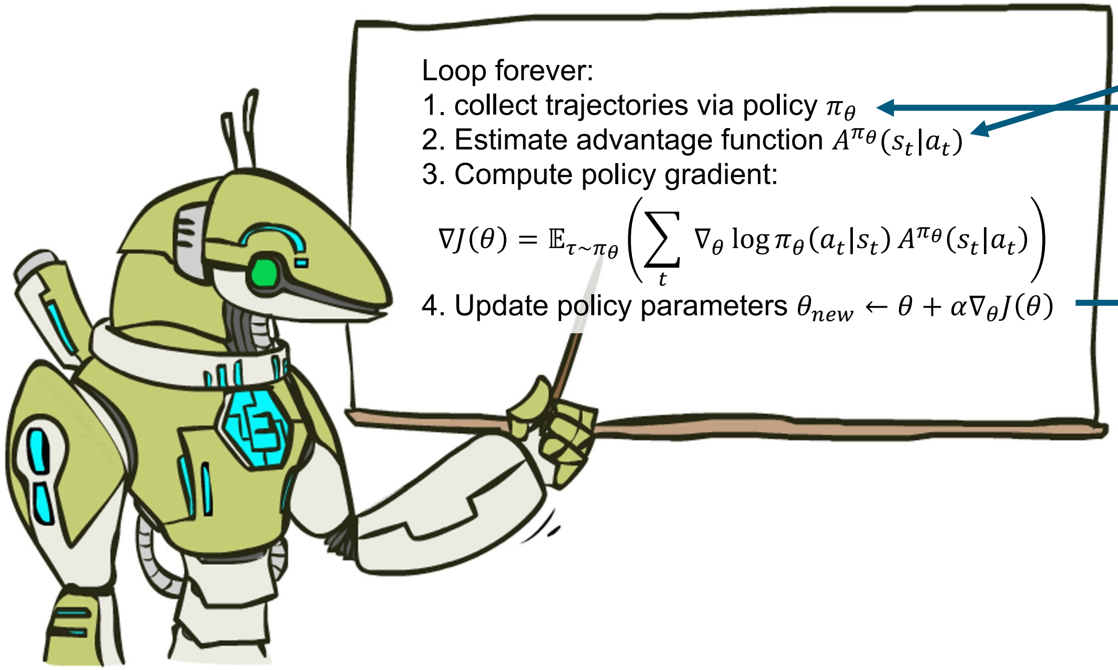
Trust-Region Policy Optimization (TRPO)

Back to RL…

→ goto Slide 55

The problem(s) of the Policy Gradient (PG) is that

- PG keeps old and new policy close in parameter space (not in policy space!), while
- small changes can lead to large differences in performance, and
- “large” step-sizes hurt performance (whatever “large” means…)



random at beginning

Non-stationary input data due to changing policy and reward distribution change

Update carefully

→ We want improvement but not degradation

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- We want to optimize $\eta(\pi)$, i.e., the expected return of policy π:

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a^t \sim \pi_{old}(\cdot|s_t)}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

- We collect data with π_{old} and optimize to get a new policy π_{new}
- In fact, so far…
  - we did not really *optimize* anything, as
  - we just favored those $(a_t|s_t)$ that had a higher advantage

- **Question:**
  - How can we write down an optimization problem that allows to do small updates on a policy π based on data sampled from π (on-policy data)?

[1] *Kakade et al.: Approximately Optimal Approximate Reinforcement Learning. ICML 2002.*

Trust-Region Policy Optimization (TRPO)

only for reference

- We want to optimize $\eta(\pi)$, i.e., the expected return of policy π:

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a^t \sim \pi_{old}(\cdot|s_t)}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

- We collect data with π_{old} and optimize to get a new policy π_{new}
- Let's express $\eta(\pi_{new})$ in terms of advantage over the original policy[1]:

$$\eta(\pi_{new}) = \eta(\pi_{old}) + \mathbb{E}_{\tau \sim \pi_{new}}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{old}}(s_t, a_t)\right]$$

Expected return of the new policy

Expected return of the old policy

Sample from new policy

[1] *Kakade et al.: Approximately Optimal Approximate Reinforcement Learning. ICML 2002.*

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- We want to optimize $\eta(\pi)$, i.e., the expected return of policy π:

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a^t \sim \pi_{old}(\cdot|s_t)}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

- We collect data with π_{old} and optimize to get a new policy π_{new}
- Let's express $\eta(\pi_{new})$ in terms of advantage over the original policy[1]:

$$\eta(\pi_{new}) = \eta(\pi_{old}) + \mathbb{E}_{\tau \sim \pi_{new}}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{old}}(s_t, a_t)\right]$$

$$= \eta(\pi_{old}) + \sum_{s} \rho_{\pi_{new}}(s) \sum_{a} \pi_{new}(a|s) A_{\pi_{old}}(s, a)$$

<u>Discounted</u> visitation frequency according to **new** policy:
$\rho_{\pi_{new}}(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots$

[1] *Schulman et al.: Trust-Region Policy Optimization. ICML 2015.*

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- We want to optimize $\eta(\pi)$, i.e., the expected return of policy π:

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a^t \sim \pi_{old}(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

- We collect data with π_{old} and optimize to get a new policy π_{new}
- Let's express $\eta(\pi_{new})$ in terms of advantage over the original policy:

$$\eta(\pi_{new}) = \eta(\pi_{old}) + \mathbb{E}_{\tau \sim \pi_{new}} \left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{old}}(s_t, a_t) \right]$$

$$= \eta(\pi_{old}) + \sum_s \rho_{\pi_{new}}(s) \sum_a \underbrace{\pi_{new}(a|s) A_{\pi_{old}}(s, a)}_{> 0}$$

new expected return > old expected return

If we can guarantee this…

→ New objective guarantees improvement from $\pi_{old} \rightarrow \pi_{new}$

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- We want to optimize $\eta(\pi)$, i.e., the expected return of policy π:

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a^t \sim \pi_{old}(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

- We collect data with π_{old} and optimize to get a new policy π_{new}
- Let's express $\eta(\pi_{new})$ in terms of advantage over the original policy:

$$\begin{aligned} \eta(\pi_{new}) &= \eta(\pi_{old}) + \mathbb{E}_{\tau \sim \pi_{new}} \left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{old}}(s_t, a_t) \right] \\ &= \eta(\pi_{old}) + \boxed{\sum_s \rho_{\pi_{new}}(s)} \sum_a \pi_{new}(a|s) A_{\pi_{old}}(s, a) \end{aligned}$$

However, this cannot be easily estimated. The state visitations that we sampled so far are coming from the old policy!

→ **we cannot optimize this in the current form!**

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

approximate locally

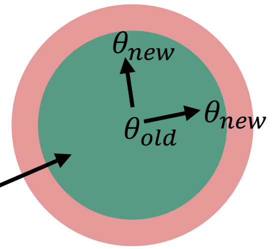
$$\eta(\pi_{new}) = \eta(\pi_{old}) + \sum_{s} \rho_{\pi_{new}}(s) \sum_{a} \pi_{new}(a|s) A_{\pi_{old}}(s,a)$$

$$\approx$$

$$L(\pi_{new}) = \eta(\pi_{old}) + \sum_{s} \rho_{\pi_{old}}(s) \sum_{a} \pi_{new}(a|s) A_{\pi_{old}}(s,a)$$

This we already sampled
→ We already have this!

- The approximation is accurate within step size δ (trust region)
  - δ needs to be chosen based on a lower-bound approximation error
- Monotonic improvement guaranteed
  - (within the green region!)



Trust region
→ $\pi_{\theta_{new}}(s,a)$ does not change too much

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- If we want to optimize $L(\theta_{new})$ instead of $\eta(\theta_{new})$ …
  with a guarantee of monotonic improvement on $\eta(\theta_{new})$, …
  … we need a bound on $L(\theta_{new})$.

- It can be proven that there exists the following bound[1,2]:

$$\eta(\pi_{new}) \geq L(\pi_{new}) - C \cdot D_{KL}^{max}\,(\pi_{old}, \pi_{new}), \text{ where } C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}$$

[1] *Schulman et al.: Trust-Region Policy Optimization. ICML 2015.*

[2] *Kakade et al.: Approximately Optimal Approximate Reinforcement Learning. ICML 2002.*

Trust-Region Policy Optimization (TRPO)

only for reference

Primer: KL-Divergence

- A measure of distance between two probability distributions P and Q:

$$D_{KL}(P\|Q) = \mathbb{E}_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

- Intuition stems from theory of channel coding:
  - The amount of **information** I (in *bits* or *nats*) that is transmitted from a sender to a receiver depends on the actual probability of the actual incident/event

$$I(x) = \log_a\left(\frac{1}{p_x}\right) = \log_\mathrm{a} 1 - \log_a p_x = -\log_\mathrm{a} p(x)$$

  - *Example #1: two events A and B have a probability of 0.75 and 0.25; then the information that A has happened is* $-log_2(0.75) = 0.41$ *(and for B it is 2)*
  - *Example #2: A and B are equally likely, i.e., 0.5 each; then the information about which of them has happened is* $-log_2(0.5) = 1$
  - If we use base 2 it tells us how much uncertainty is cut off by half!

- For simplification we use nats as $\log_2 x = \log x/\log 2$ ($\log 2$ is constant)

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

Primer: KL-Divergence

- The **entropy** (of a probability distribution) is given by:

$$H(p) = -\sum_i p_i \log(p_i)$$

  - It measures the average amount of information from one sample drawn from the underlying probability distribution p
    → it measures how unpredictable p is
    → defines the lower bound of the optimal bit-encoding

- As usual, we can also write $H(p)$ in its expectation and draw x from p:

$$H(p) = \mathbb{E}_{x \sim p}[-\log p(x)]$$

Trust-Region Policy Optimization (TRPO)

Primer: KL-Divergence

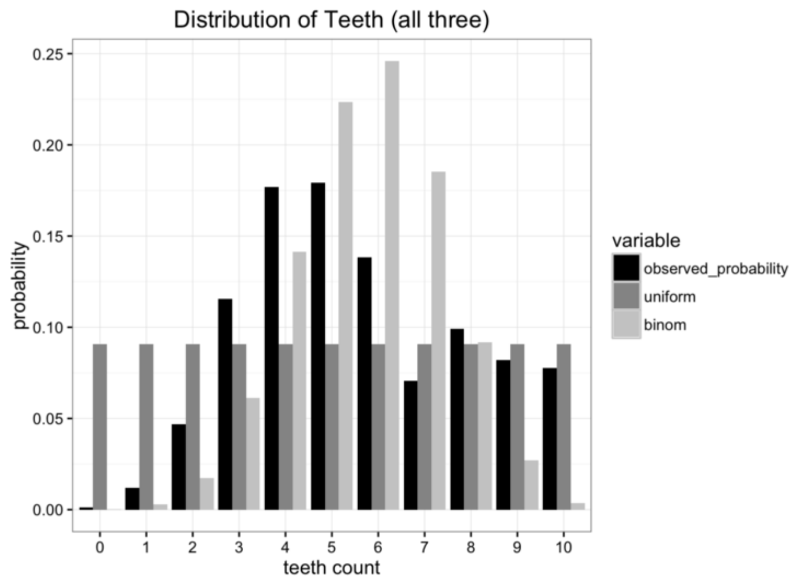
- The **cross-entropy** defines the average number of bits to identify a sampled event if it is encoded using q rather than p:

$$H(p,q) = -\sum_i p_i \log_2(q_i)$$

  - tells us the difference about the true probability distribution p and the predicted probability distribution q (given the encoding)
  - If p and q are equal, then the cross-entropy and the entropy are equal

- As before, we can write $H(p,q)$ in its expectation:

$$H(p,q) = \mathbb{E}_{x\sim p}[-\log q(x)]$$



from https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

Primer: KL-Divergence

- **KL-Divergence** tells us how much far away the cross-entropy is from the entropy (note that $H(p,q) \geq H(p)$) :

$$D_{KL}(p||q) = H(p,q) - H(p)$$

- Measure of dissimilarity between two probability distributions P and Q:

$$\begin{aligned} D_{KL}(p||q) &= \mathbb{E}_{x\sim p}[-\log q(x)] - \mathbb{E}_{x\sim p}[-\log p(x)] \\ &= \mathbb{E}_{x\sim p}[-\log q(x) - (-\log p(x))] \\ &= \mathbb{E}_{x\sim p}[-\log q(x) + \log p(x)] \\ &= \mathbb{E}_{x\sim p}[\log p(x) - \log q(x)] = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right] \end{aligned}$$

$D_{KL}(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$ and $D_{KL}(p||q) = \int P(x) \log \frac{p(x)}{q(x)} dx$

- *Note: KL-Divergence is asymmetric and hence no distance metric!*

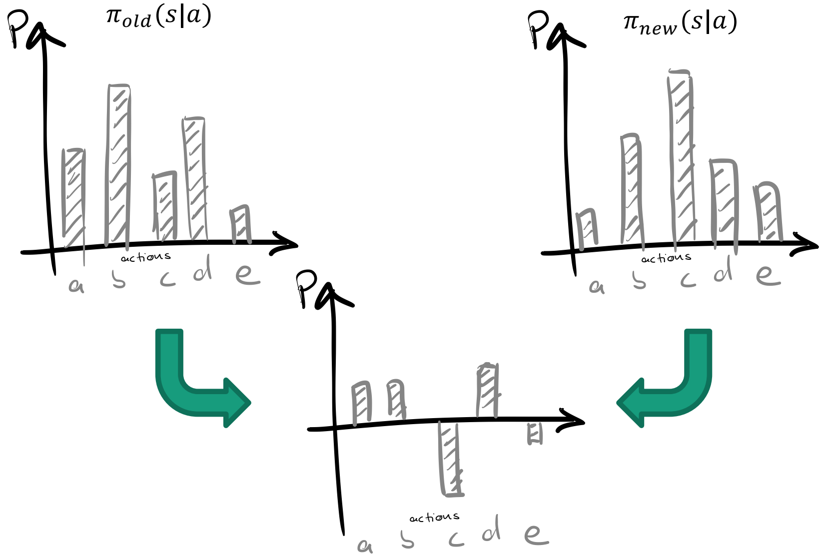
for more information see also https://www.youtube.com/watch?v=ErfnhcEV1O8 and https://medium.com/activating-robotic-minds/demystifying-kl-divergence-7ebe4317ee68

Trust-Region Policy Optimization (TRPO)

Back to RL…

- If we want to optimize $L(\theta_{new})$ instead of $\eta(\theta_{new})$ …
  with a guarantee of monotonic improvement on $\eta(\theta_{new})$, …
  … we need a bound on $L(\theta_{new})$.
- It can be proven that the following bound holds[1,2]:

$$\eta(\pi_{new}) \geq L(\pi_{new}) - C \cdot D_{KL}^{max}(\pi_{old}, \pi_{new}), \text{ where } C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}$$



$$D_{KL}(\pi_{old} \| \pi_{new}) = \pi(a) \log \frac{\pi_{old}(a)}{\pi_{new}(a)}$$

[1] Schulman et al.: Trust-Region Policy Optimization. ICML 2015.
[2] Kakade et al.: Approximately Optimal Approximate Reinforcement Learning. ICML 2002.

Trust-Region Policy Optimization (TRPO)

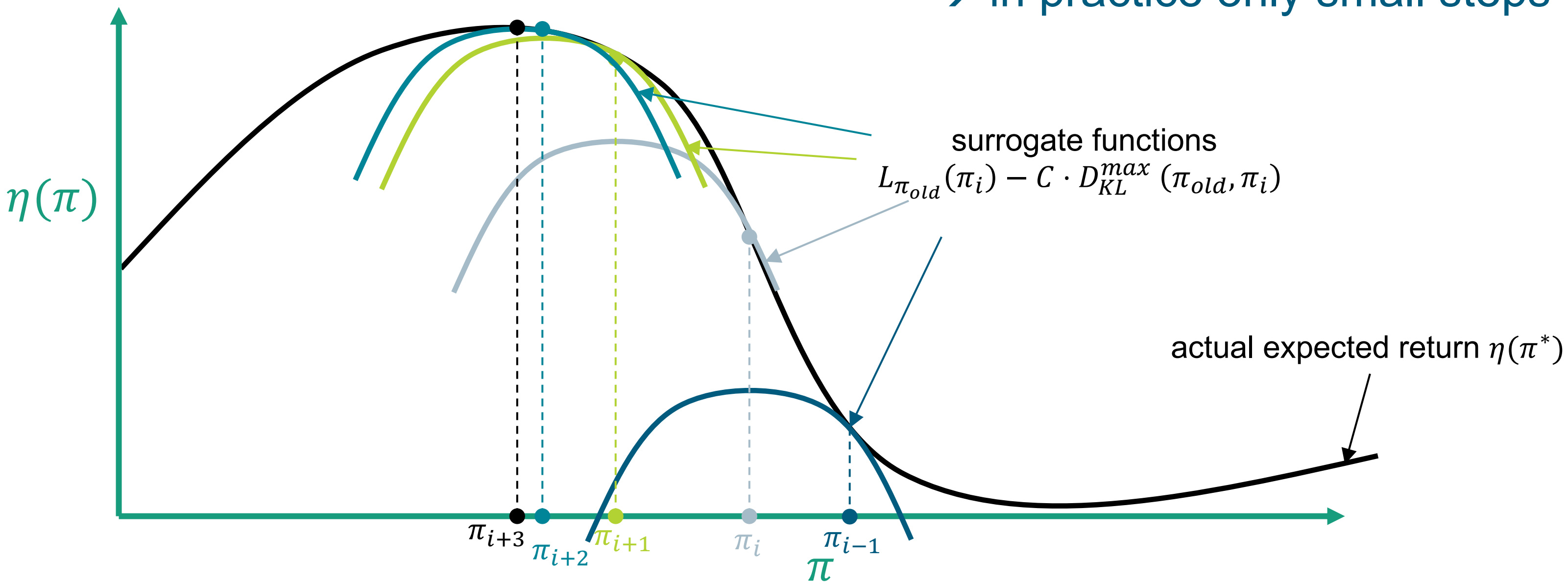
Minorization-Maximization Algorithm

→ goto Slide 63

- A monotonically increasing policy can be defined by:

$$\pi = \arg\max_{\pi}[L(\pi_{new}) - C \cdot D_{KL}^{max}(\pi_{old}, \pi_{new})], \text{ where } \boxed{C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}}$$

Hard to tune
→ in practice only small steps



Trust-Region Policy Optimization (TRPO)

Minorization-Maximization Algorithm

- A monotonically increasing policy can be defined by:

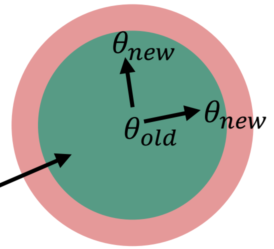
$$\pi = \arg\max_{\pi}[L(\pi_{new}) - C \cdot D_{KL}^{max}(\pi_{old}, \pi_{new})], \text{ where } C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}$$

Side-note:

- A constraint on the KL-divergence between new and old policy (i.e., a trust region constraint) allows larger step sizes while being mathematically equivalent:

$$\pi = \arg\max_{\pi} L_{\pi_{old}}, \text{ such that } D_{KL}^{max}(\pi_{old}, \pi) \leq \delta$$

- Approximation with L is accurate within δ
  → here, monotonic improvement guaranteed



Trust region
→ $\pi_{\theta_{new}}(s, a)$ does not change too much

Trust-Region Policy Optimization (TRPO)

only for reference

- Solving the KL-penalized problem (btw: directly over θ not π_θ)

$$\arg\max_\theta \left[L(\tilde{\theta}) - C \cdot D_{KL}^{max}(\theta, \theta_{old})\right]$$

- Use mean KL-divergence instead of max:

$$\arg\max_\theta \left[L(\tilde{\theta}) - C \cdot \overline{D_{KL}}(\theta, \theta_{old})\right]$$

linear approximation of L

quadratic approximation of KL

$$\arg\max_\theta \left[\frac{\partial}{\partial\theta} L(\theta)\Big|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) - \frac{C}{2}(\theta - \theta_{old})^T \cdot \frac{\partial^2}{\partial^2\theta}\overline{D_{KL}}(\theta, \theta_{old})\Big|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old})\right]$$

$$\arg\max_\theta \left[g \cdot (\theta - \theta_{old}) - \frac{C}{2}(\theta - \theta_{old})^T F (\theta - \theta_{old})\right]$$

$$g = \frac{\partial}{\partial\theta} L(\theta)\Big|_{\theta=\theta_{old}} \qquad F = \frac{\partial^2}{\partial^2\theta}\overline{D_{KL}}(\theta, \theta_{old})\Big|_{\theta=\theta_{old}}$$

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

only for reference

- Derive with respect to θ and set to 0:

$$0 = \frac{\partial}{\partial\theta}\left[g \cdot (\theta - \theta_{old}) - \frac{c}{2}(\theta - \theta_{old})^T F(\theta - \theta_{old})\right]$$
$$0 = 1 \cdot \left[g - \frac{c}{2}(\theta - \theta_{old})^T F\right] + (\theta - \theta_{old})\left(-\frac{c}{2}F\right)$$
$$\mathrm{g} - \frac{c}{2}(\theta - \theta_{old})^T F = \frac{c}{2}(\theta - \theta_{old})F$$
$$F^{-1}g - \frac{c}{2}(\theta - \theta_{old})^T = \frac{c}{2}(\theta - \theta_{old})$$
$$F^{-1}g = c(\theta - \theta_{old})$$
$$\frac{1}{c}F^{-1}g = \theta - \theta_{old}$$

- Solution: iterative optimization with Newton‘s method:

$$\theta_{new} = \theta_{old} + \frac{1}{c}H^{-1}g$$

Trust-Region Policy Optimization (TRPO)

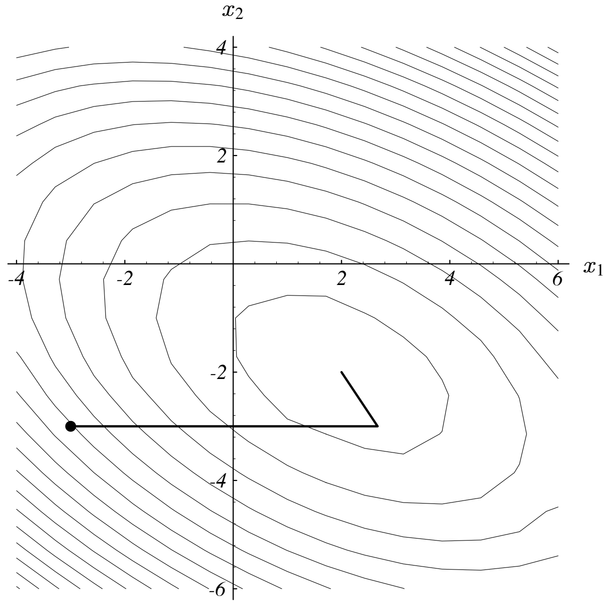
only for reference

- The update step

$$\theta_{new} = \theta_{old} + \frac{1}{c} H^{-1} g$$

  involves computing the inverse of the Hessian
  → too expensive for large $|\theta|$

- Conjugate Gradient (CG) computes $F^{-1}g$ approximately without forming F explicitely
  - CG solves $x = A^{-1}b$ without explicitely forming A
  - After k iterations, CG has minimized $\frac{1}{2}x^T Ax - bx$



Jonathan Richard Shewchuk: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Edition 1 ¼. 1994.

Trust-Region Policy Optimization (TRPO)

only for reference

- Unconstrained problem: $\arg\max_{\theta} L(\theta) - C \cdot \overline{D_{KL}}(\theta_{old}, \theta)$
- Constrained problem: $\arg\max_{\theta} L(\theta)$ subject to $C \cdot \overline{D_{KL}}(\theta, \theta_{old}) \leq \delta$
  - δ is a hyper-parameter, remains fixed over the whole learning process

- Solve constrained quadratic problem: compute $F^{-1}g$ and then rescale step to get correct KL:
  1. $\max_{\theta} g \cdot (\theta - \theta_{old})$ subject to $\frac{1}{2}(\theta - \theta_{old})^T F)(\theta - \theta_{old}) \leq \delta$
  2. Lagrangian: $\mathcal{L}(\theta, \lambda) = g \cdot (\theta - \theta_{old}) - \frac{c}{2}[(\theta - \theta_{old})^T F(\theta - \theta_{old}) - \delta]$
  3. Differentiate with respect to θ and get $\theta - \theta_{old} = \frac{1}{c} F^{-1} g$
  4. We want $\frac{1}{2} s^T F s = \delta$
  5. Given candidate step $s_{unscaled}$ rescale $s = \sqrt{\frac{2\delta}{s_{unscaled} \cdot F \cdot s_{unscaled}}} \cdot s_{unscaled}$

Trust-Region Policy Optimization (TRPO)

only for reference

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

You have seen this very often so far

Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

This is an approximation to the *natural policy gradient (2nd order)*

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

You have seen this very often so far

end for

Fraunhofer IIS

Trust-Region Policy Optimization (TRPO)

Intuition & Summary

- We approximate the expected return function locally around the current policy.
  - The accuracy decreases when the new policy and the current policy diverge from each other.

- But we can establish an upper bound for the error.
  - Therefore, we can guarantee a policy improvement if we optimize the local approximation within a trusted region.
  - Outside this region, the bet is off.

- Even it may have a better-calculated value, its range of error fails the improvement guarantee. With such a guarantee inside the trust region, we can locate the optimal policy iteratively. So even it takes a while to prove it mathematically, the reasoning is simple.

Trust-Region Policy Optimization (TRPO)

Intuition & Summary

- Shortcomings:
  - TRPO minimizes a quadratic equation to approximate the inverse of the Fisher Information Matrix (FIM), i.e., the Hessian
  - To do this for every policy is expensive
  - It requires a large batch of rollouts to approximate it correctly
  - Less sample-efficient to other PG methods when trained with first-order methods such as Adam

→ These become a significant issue for large/deep networks!

- **And: Yes, TRPO is a complex algorithm that is hard to understand & implement**

Fraunhofer IIS

Policy-based RL
Agenda

- Policy-based RL 1
  - Intro to Policy-based RL
  - Policy Gradients

- Policy-based RL 2
  - Variance & Baselines
  - Actor-Critics
  - Trust-Region Policy Optimization (TRPO)
  - **Proximal Policy Optimization (PPO)**
  - Deep Deterministic Policy Gradient (DDPG)

Proximal Policy Optimization (PPO)

- The main motivation behind PPO is the same as for TRPO:
  - Make the biggest possible improvement step
  - Do not step too far such that the performance accidentally collapses

- PPO addresses the shortcomings of TRPO:
  - PPO uses 1^{st} order methods with a few tricks
  - Significantly simpler to implement
  - Shows similar performance to TRPO (empirically)

- There are two variants:
  - PPO-penalty: TRPO with KL-penalization instead of constraint (penalty coefficient is adjusted and scaled automatically over the course of training: *Adaptive KL Penalty Coefficent*)
  - PPO-clip: no constraints! Adds a clipping to the objective function to remove incentives to move too far

Spoiler: PPO is (1) much simpler to understand and to implement, and (2) much better (empirically)

Proximal Policy Optimization (PPO)

Where are we so far?

- TRPO maximizes a "surrogate" objective subject to a constraint on the size of the policy update:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right], \qquad \text{subject to} \quad \hat{\mathbb{E}}_t \left[KL\left[\pi_{\theta_{old}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\right]\right] \leq \delta$$

  with θ_{old} being the policy parameters before the update

- We did not explicitly formulate it like this, but the intuition behind it is:
  - We want to measure how π_θ performs relative to $\pi_{\theta_{old}}$ (using data from the old policy)
  - The original objective (see TRPO slides) can be exactly reformulated to this one
- We can solve this with CG after making a linear approximation to the objective and a quadratic approximation to the KL-constraint

Proximal Policy Optimization (PPO)

Where are we so far?

- TRPO maximizes a "surrogate" objective subject to a constraint on the size of the policy update:

$$\max_\theta \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right], \qquad \text{subject to} \quad \hat{\mathbb{E}}_t \left[KL[\pi_{\theta_{old}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)] \right] \leq \delta$$

  with θ_{old} being the policy parameters before the update

- Let us define the probability ratio $r_t(\theta)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\,old}(a_t|s_t)} \qquad \text{, i.e., } r_t(\theta_{old}) = 1.$$

- In other words, TRPO maximizes the following objective:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

  penalizing changes to the policy that move $r_t(\theta)$ (too far) away from 1.

Proximal Policy Optimization (PPO)

- The PPO objective we want to maximize is given by

$$= g\left(\epsilon, \hat{A}_t(s,a)\right) \qquad g(\epsilon, \hat{A}_t) = \begin{cases}(1+\epsilon)A, \text{if } A \geq 0 \\ (1-\epsilon)A, \text{if } A < 0\end{cases}$$

$$L(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \boxed{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]}$$

where ϵ is a hyperparameter (i.e., 0.1 or 0.2) that defines how far π_{new} may go away from π_{old}

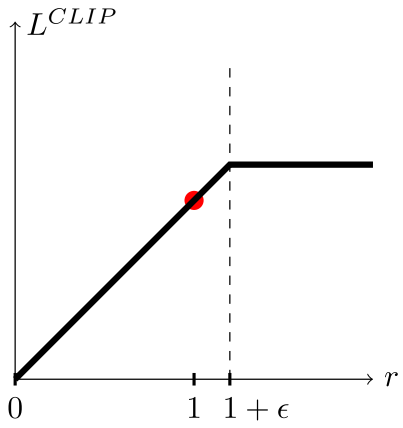
- First term inside the $\min$ is $L^{CPI}(\theta)$
- Second term inside the $\min$ clips the probability ratio
→ removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$
- We take the minimum of the clipped and unclipped objective
→ the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective

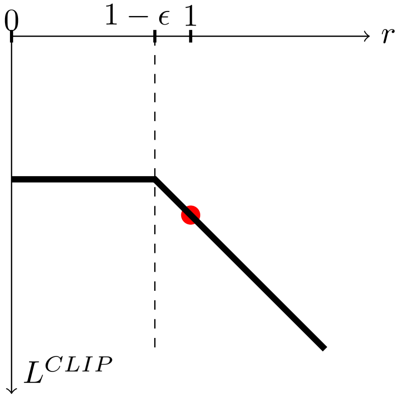
See also: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Proximal Policy Optimization (PPO)

- The clipping operator is a *pessimistic bound* of the unclipped objective

$A > 0$





$A < 0$

- Plot show a single timestep of the surrogate function L^{CLIP} as a function of r
- The red circle shows the starting point for the optimization, i.e., $r = 1$

See also: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Proximal Policy Optimization (PPO)

- Automated Tuning of the gradient step *without calculating the Hessian*

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\boxed{\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),}$$

Advantage Clipping for conservative policy updates

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

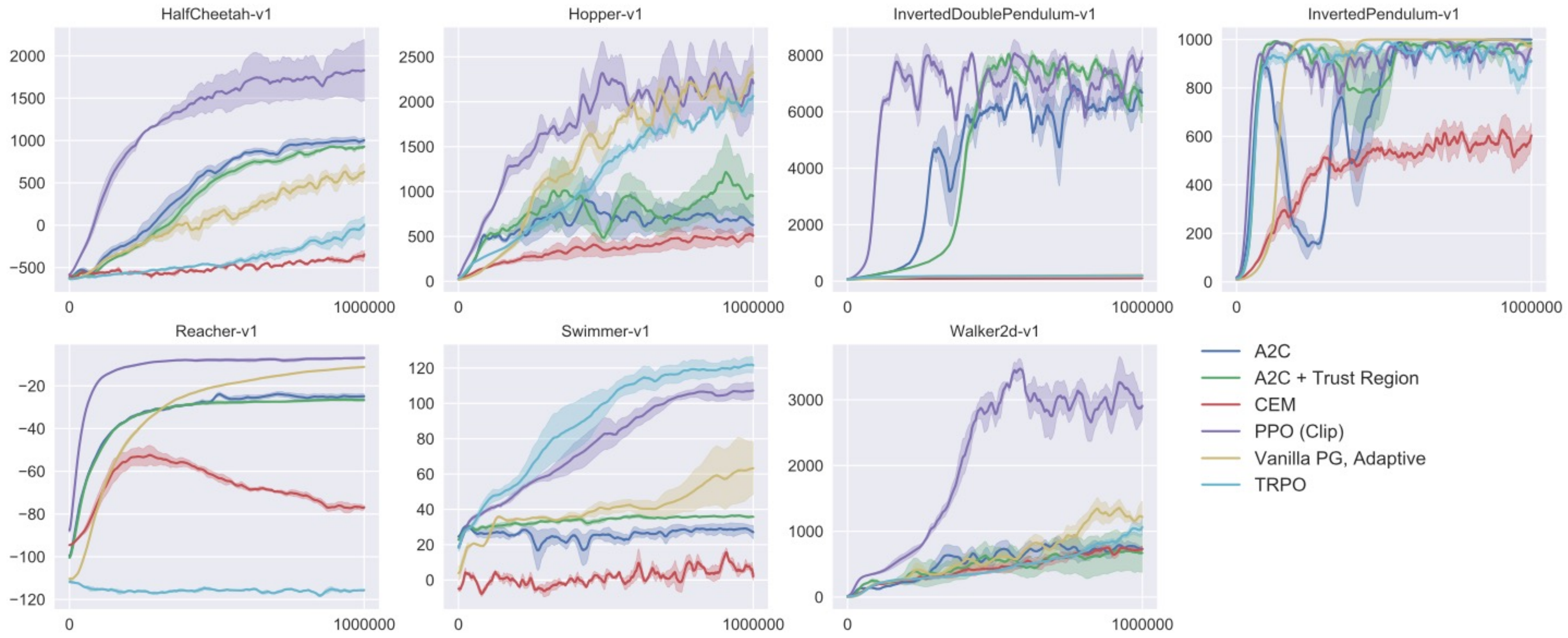
typically via some gradient descent algorithm.
8: **end for**

See also: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Fraunhofer IIS

Proximal Policy Optimization (PPO)

- Results of PPO-clip:
  - Against well-known competitors
  - On well-known environments
- Those results are impressive!



Proximal Policy Optimization (PPO)

Practical Considerations

Algorithm 1 PPO, Actor-Critic Style

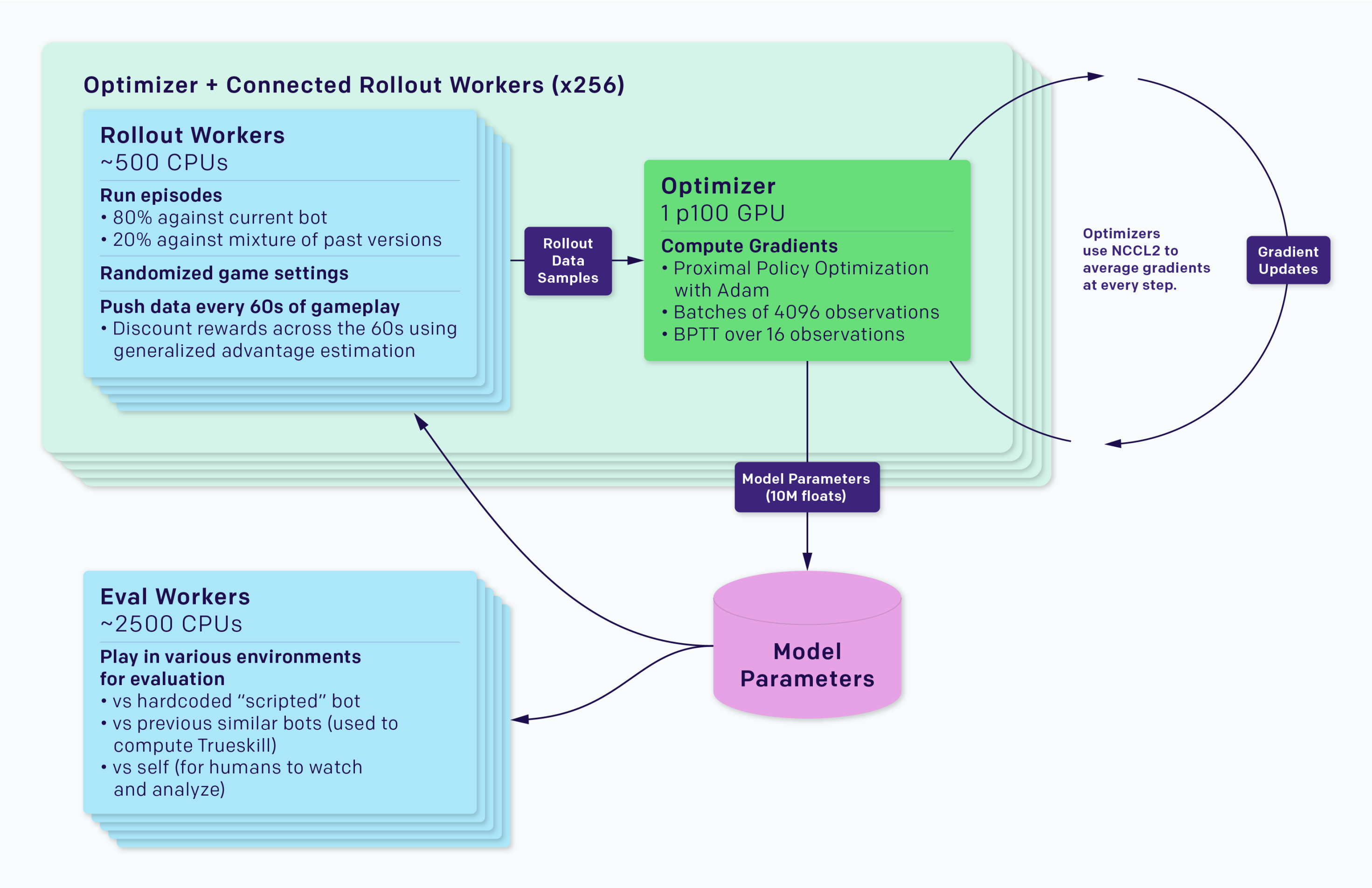
---

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

- There is two alternating threads in PPO:
  1. Policy interacts with the environment, collects data and computes advantage estimates (using fitted baselines estimates)
  2. 2nd thread collects all the experiences and runs SGD to optimize the policy using the clipped objective

Proximal Policy Optimization (PPO)

PPO in Action: OpenAI Five on DOTA II



https://openai.com/blog/openai-five/

Proximal Policy Optimization (PPO)
Practical Considerations

- One more thing….

- PPO combines a few more things in the final objective:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

clipped surrogate objective

Sqared error loss for $\left(V_\theta(s_t) - V_t^{targ}\right)^2$

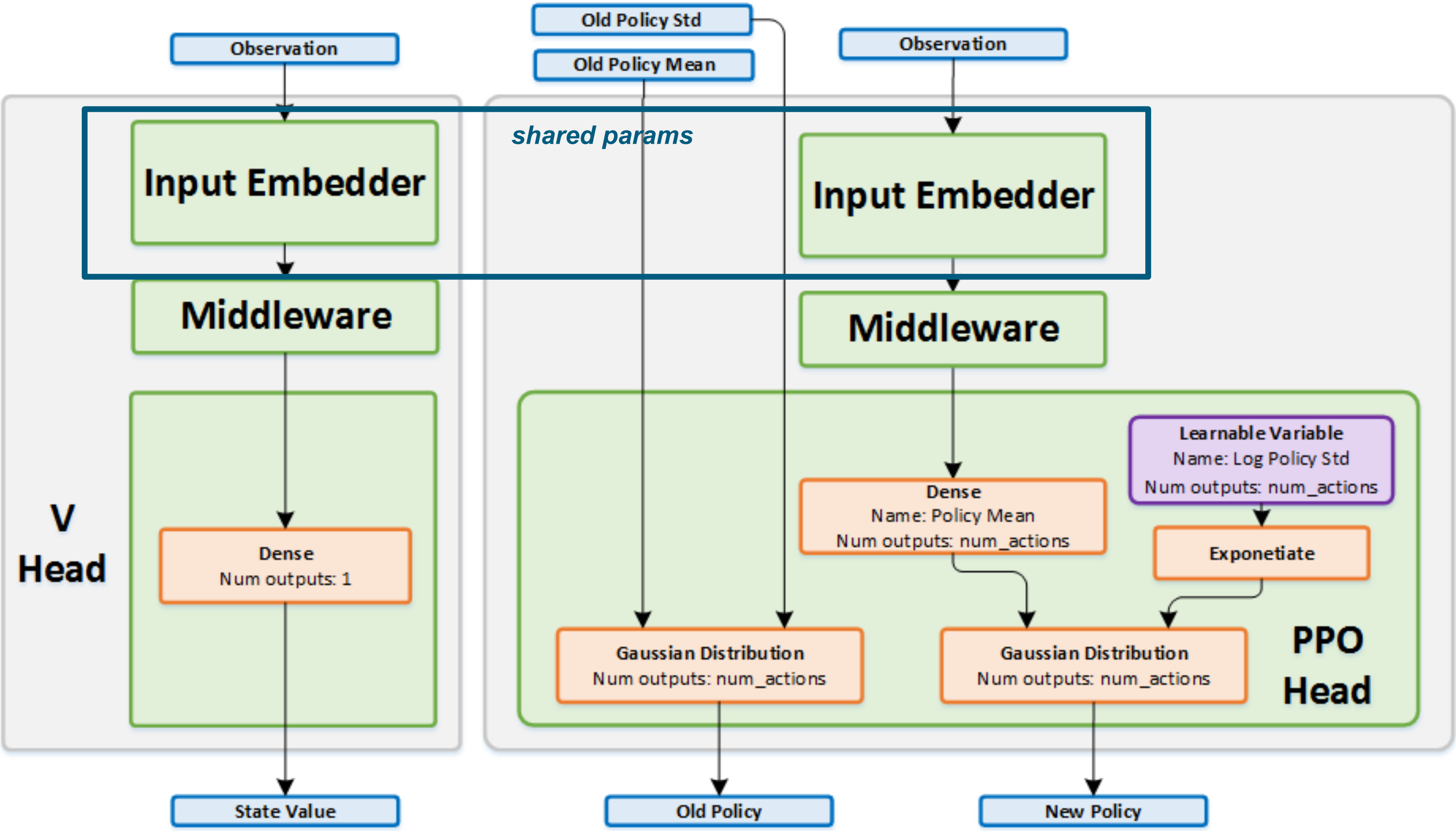
entropy bonus (exploration)

- *Note: you likely need similar features to represent the policy and the state-values*
  → OpenAI Five shares parameters between the policy network and the value network
  → Both error terms are combined in a single loss function

Fraunhofer IIS

Proximal Policy Optimization (PPO)

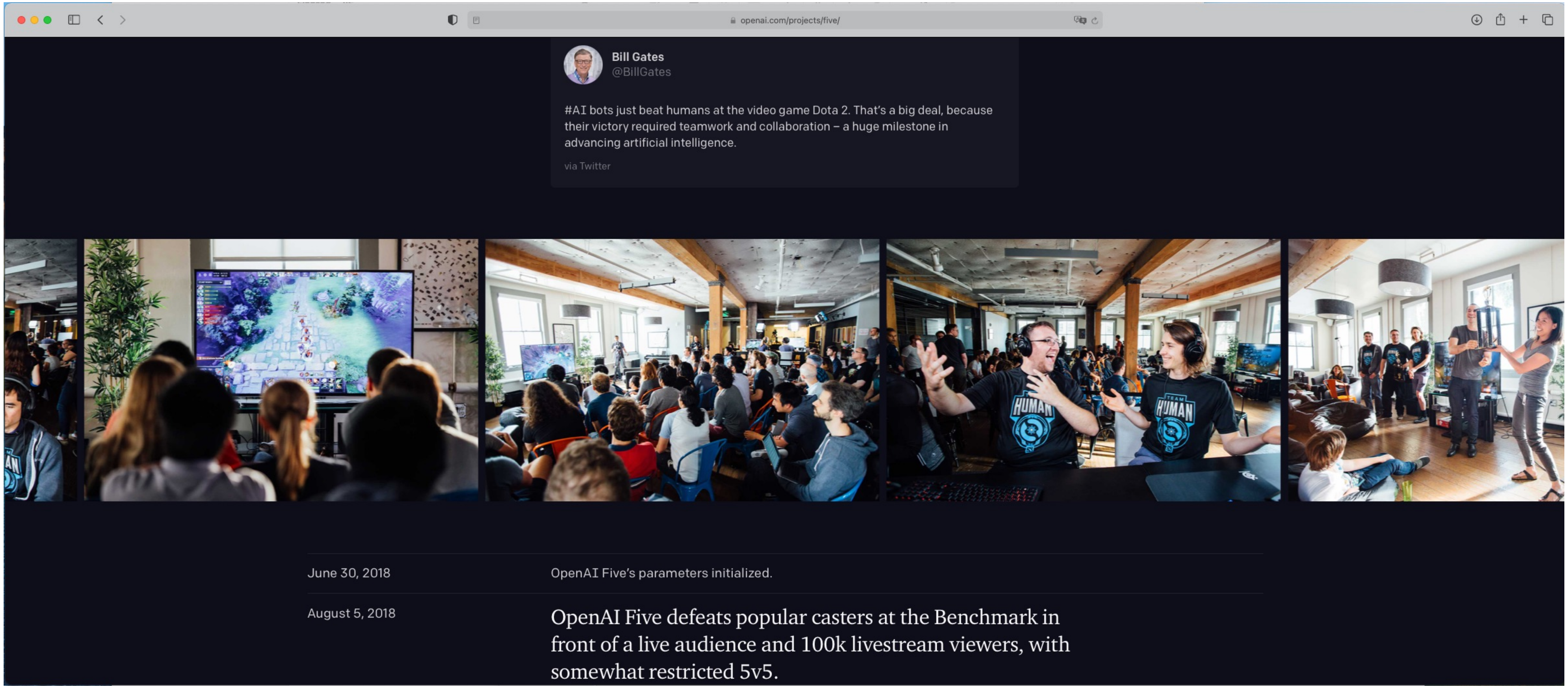
PPO in Action: OpenAI Five vs. DOTA II



https://intellabs.github.io/coach/components/agents/policy_optimization/ppo.html

Proximal Policy Optimization (PPO)

PPO in Action: OpenAI Five vs. DOTA II



https://openai.com/projects/five/

Fraunhofer IIS

Proximal Policy Optimization (PPO)

PPO in Action: OpenAI Five vs. DOTA II

- **High-level info:**
  - 180 years of self-play per day and hero (~900 yrs/day), no human data
  - Running on 256 P100 GPUs and 128,000 CPU cores
- **Technical stats:**
  - Observation size: ~36.8kB @ ~7Hz
  - Batch size: 1,048,576 observations = 36 GB ☺
  - Separate single-layer, 1024 unit LSTM per hero
  - See https://openai.com/blog/openai-five/ for an interactive demo!
  - Reward: net worth, kills, deaths, assist, last hits, etc.
  - "Team spirit" – trade own rewards over team reward (heroes do not communicate)

- **Challenge: exploring combinatorial-vast space of combining actions w/ long planning horizons**
  - 80% of games against itself, 20% against past selves (avoid "strategy collapse")
  - After several hours: concepts such as laning, farming or fighting emerged
  - After several days: basic human strategies such as steal bounty runes from opponents, rotate heroes around the map to gain lane advantage etc.

Policy-based RL

Agenda

- Policy-based RL 1
  - Intro to Policy-based RL
  - Policy Gradients

- Policy-based RL 2
  - Variance & Baselines
  - Actor-Critics
  - Trust-Region Policy Optimization (TRPO)
  - Proximal Policy Optimization (PPO)
  - **Deep Deterministic Policy Gradient (DDPG)**

Deep Deterministic Policy Gradient (DDPG)

Question: can we also use ideas from value-based RL for continuous action spaces?

- Why can't we use Q-Learning and DQNs?

→ **Not so easily!**

But what was the original problem with Q-Learning?

- Q-Learning and variants (including DQNs) do not work with continuous actions
- Why is that? Remember:
  - We calculated the targets $r_t + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \theta_{i-1})$ by a single pass through the network
  - Our network was "static" and had $|\mathcal{A}|$ outputs
- Evaluating a continuous action space requires an exhaustive search over the available actions (and this becomes highly non-trivial!)[1]

[1] *However, see approaches such as Lim et al.: Actor-Expert: A Framework for using Q-learning in Continuous Action Spaces.*

Deep Deterministic Policy Gradient (DDPG)

- DDPG learns a Q-function and a policy
  - Off-policy data + Bellman equation to learn Q-function
  - Make use of the Q-function to learn the policy

- The intuition relies on Q-learning:
  - If you know $Q^*(s, a)$ then in each state the optimal action $a^*(s)$ can be *simply* found by

$$a^*(s) = \arg\max_a Q^*(s, a)$$

  - DDPG jointly learns approximations to $Q^*(s, a)$ and $a^*(s)$

    …and specifically adapts this for continuous action spaces!

Deep Deterministic Policy Gradient (DDPG)

Main idea

- We assume the function $Q^*(s, a)$ to be differentiable with respect to a
- This allows for a gradient-based learning rule for a policy $\mu(s)$ that exploits this

- Instead of exhaustively looking for $\max_a Q(s, a)$ we approximate it:

$$\max_a Q(s, a) \approx Q(s, \mu(s))$$

- We look at two sides of DDPG:
  1. Its Q-Learning side
  2. Its policy gradient side

Deep Deterministic Policy Gradient (DDPG)

The Q-Learning side of DDPG

- Recap the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}\left[r(s,a) + \gamma \max_a Q^*(s',a')\right]$$

- Then we can optimize the mean-squared Bellman error (MSBE) (neural network parameters ϕ, set of transitions D, $d \in \{0;1\}$ indicates if s' is terminal):

$$L(\phi, D) = \mathbb{E}_{(s,a,r,s',d))\sim D}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

- For optimization using SGD we apply the well-known tricks:
  - Replay buffers (off-policy!)
  - Use target networks and update it with delay by $\phi_{targ} \leftarrow p\phi_{targ} + (1-p)\phi, \;\; p \in [0;1]$

Deep Deterministic Policy Gradient (DDPG)

The Q-Learning side of DDPG

- Calculating the max over the actions in the target:
  - Target policy network μ_θ computes an action that approximately maximizes $Q_{\phi_{targ}}$
  - Target policy updates also computed using polyak averaging (see above)
  - Q-Learning in DDPG minimizes using SGD:

$$L(\phi, D) = \mathbb{E}_{(s,a,r,s',d)\sim D}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{targ}}\left(s', \mu_{\theta_{targ}}(s')\right)\right)\right)^2\right]$$

Deep Deterministic Policy Gradient (DDPG)

The Policy Learning side of DDPG: simple

- Policy $\mu_\theta(s)$ is deterministic
- $\mu_\theta(s)$ should return the action that maximizes $Q_\phi(s,a)$
- Action space is continuous, and we assume Q to be differentiable with respect to actions a

- Hence, we can use gradient ascent (with respect to the policy parameters θ only) and solve:

$$\max_\theta \mathbb{E}_{s\sim D}\left[Q_\phi(s,\mu_\theta(s))\right]$$

(Q-function parameters ϕ are treated as constants here)

Deep Deterministic Policy Gradient (DDPG)

Exploration-Exploitation

- DDPG trains off-policy; policy is deterministic
- Hence an on-policy exploration is often not enough
- Solution: add noise to the actions at training time
  - Originally time-correlated Ornstein-Uhlenbeck (OU) process noise has been proposed
  - More recent work suggests to use zero-mean Gaussian noise as it is simpler and exhibits same performance
  - Over the course of training, we may reduce the scale of noise (but there is only a limited effect from this)
  - At test time we (of course) omit to add the noise

Deep Deterministic Policy Gradient (DDPG)

Pseudo Code

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

Use target networks (as in DQN) for both the actor and the critic

As we don't have a stochastic policy, we have to define a process for exploration

Execute action, store transition in the replay buffer and sample at random some transitions (exactly as in DQN)

Update critic using the target networks for both the Actor and the Critic

Update the Actor using the Deterministic Policy Gradient Theorem (Silver et al. 2014)

Progressively update the target networks

Deep Deterministic Policy Gradient (DDPG)

Summary

- Special case in actor-critic-algorithms:
- Works only for continuous action spaces
- Is an off-policy algorithm utilizing the replay buffer trick from DQN
- Solves for deterministic policies instead of stochastic ones

\+ impressive results both in simulation and in real world problems

\+ one of the de-facto algorithms to use for continuous (or very large) action spaces

\- very sensitive to the exploration process

\- can be hard to tune – sensitive to hyperparameters

\- slower (wall clock time) compared to other actor-critic-algorithms

References

Policy-based Reinforcement Learning:

- Deisenroth, M. P., Neumann, G., & Peters, J. (2013). A survey on policy search for robotics. Foundations and Trends® in Robotics, 2(1–2), 1-142: https://spiral.imperial.ac.uk/bitstream/10044/1/12051/7/fnt_corrected_2014-8-22.pdf
- Sigaud, O., & Stulp, F. (2019). Policy search in continuous action domains: an overview. Neural Networks. ArXiv: https://arxiv.org/pdf/1803.04706.pdf
- Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems (pp. 1057-1063). Link: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf
- Kakade, S. M. (2002). A natural policy gradient. In Advances in neural information processing systems (pp. 1531-1538). Link: http://papers.nips.cc/paper/2073-a-natural-policy-gradient.pdf
- Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016, June). Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning (pp. 1329-1338): http://proceedings.mlr.press/v48/duan16.pdf
- Riedmiller, M., Peters, J., & Schaal, S. (2007, April). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (pp. 254-261). IEEE. link: http://is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/ADPRL2007-Peters2_[0].pdf
- Kober, J., & Peters, J. R. (2009). Policy search for motor primitives in robotics. In Advances in neural information processing systems (pp. 849-856). Link: https://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4), 229-256.
- Vlassis, N., Toussaint, M., Kontes, G., & Piperidis, S. (2009). Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots, 27(2), 123-130.

References

Actor-Critics:

- Deep RL Bootcamp 2017:
  - Pieter Abbeel: Policy Gradients (Lecture 4A), Aug. 26th, 2017
  - Andrej Karpathy: Pong from Pixels (Lecture 4b), Aug. 26th, 2017
- https://www.janisklaise.com/post/rl-policy-gradients/
- OpenAI Spinning Up: "Vanilla Policy Gradient". https://spinningup.openai.com/en/latest/algorithms/vpg.html
- OpenAI Spinning Up: "Intro to Policy Optimization". https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#other-forms-of-the-policy-gradient
- Williams, Ronald J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning 8:229-256.

Trust-Region Policy Optimization (TRPO):

- Pascal Poupart: CS885 Lecture 14c: Trust Region Methods
- Sergey Levine: CS285: Advanced Policy Gradients
- In-depth Research Paper Review: https://www.youtube.com/watch?v=CKaN5PgkSBc