Maximum A Posteriori Policy Optimisation

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., & Riedmiller, M. (2018). Maximum a posteriori policy optimisation.

preprint: arXiv:1806.06920 (2018).

Presented by: Raeid Saqur

STA4273 L7: Policy Optimization I
March 4, 2021

MPO: Key Idea & Contributions

- *Novelty*: Introduces a novel off-policy RL algorithm using expectation maximization (EM) for optimization

- *Paradigm*: Uses '**Policy as inference**' for optimization:
  - Instead of: *What actions maximize future rewards?*,
  - Asks: <u>A*ssuming future success*</u> *in maximizing rewards,* <u>*what are the actions*</u> *most likely to have been taken*?

- *Performance*: SOTA robust, sample efficient performance on a wide range of discrete and continuous controls tasks (MuJoCo, DM-Suite [1, 2])

Background & Preliminaries

Expectation Maximization (EM) Algorithm

- If we know how to model $p(x \mid z)$ - where x, z are observed and latent variables respectively, with joint distribution $p(x, z \mid \theta)$ governed by params θ:
  - Direct optimization (e.g., MLE) can be applied: $z^* = \arg\max_z p(x \mid z; \theta)$
  - Optimize log-likelihood of *x* over all possibilities of *z* (complete data, D):

$$\log p(D) = \sum_{x \in D} \log p(x) = \underbrace{\sum_{x \in D} \log \left(\sum_z p(x \mid z) p(z) \right)}_{\text{Log of sum (intractable)}}$$

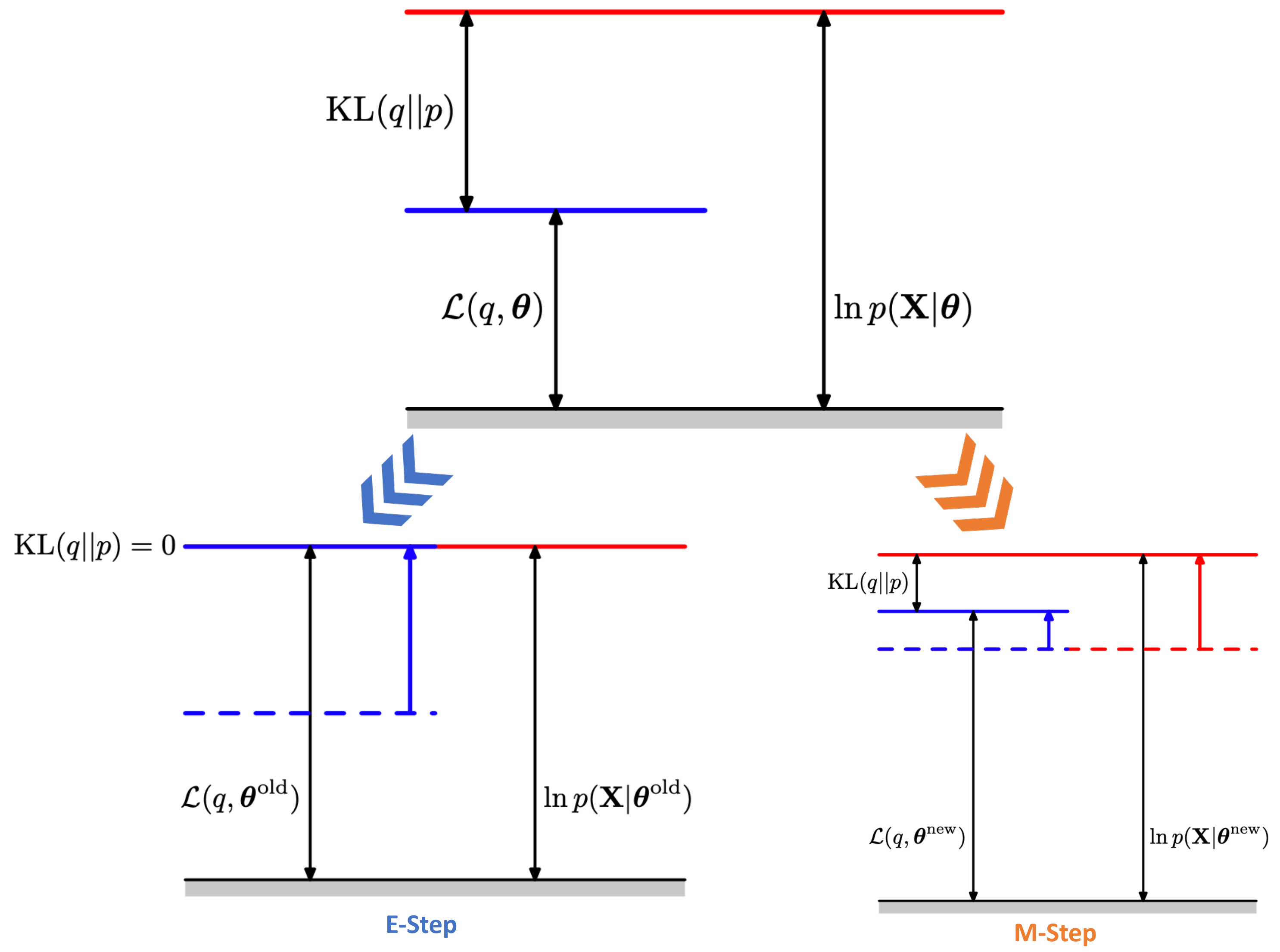
- Use EM when direct optimization $p(x|\theta)$ is difficult, but optimizing complete data likelihood $p(x, z|\theta)$ is easier. Introduce auxiliary $q(z)$ over *z* and decompose:

$$\ln p(\mathbf{X} \mid \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q \| p)$$

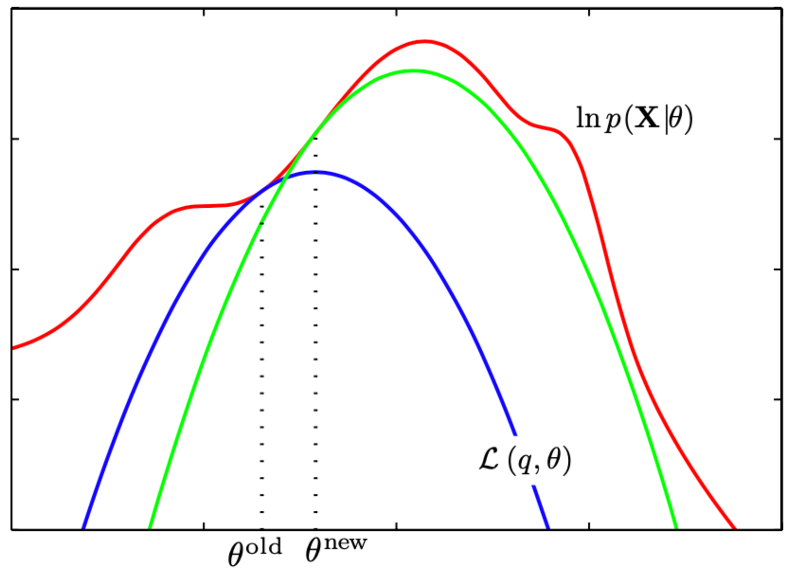
where,

$$\begin{aligned} \mathcal{L}(q, \boldsymbol{\theta}) &= \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \\ \mathrm{KL}(q \| p) &= -\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \end{aligned}$$

Background: EM Algorithm



$\mathrm{KL}(q||p)$
$\mathcal{L}(q, \boldsymbol{\theta})$
$\ln p(\mathbf{X}|\boldsymbol{\theta})$
$\mathrm{KL}(q||p) = 0$
$\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{old}})$
E-Step
$\mathrm{KL}(q||p)$
$\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{new}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{new}})$
M-Step



The EM algorithm involves alternately computing a lower bound on the log likelihood for the current parameter values and then maximizing this bound to obtain the new parameter values.

*Source: Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006.

MPO: Setup as an EM Problem

- Introduces an auxiliary distribution over trajectories: $q(\tau)$
- Define the ELBO on the likelihood of optimality for policy Π as:

$$\log p_\pi\underbrace{(O=1)}_{\text{Goal achieved}} = \log \int \overset{\substack{\text{Trajectory induced by } \pi(a|s)\\ \downarrow}}{p_\pi(\tau)} p(O=1 \mid \tau)\,d\tau \geq \int \overset{\substack{\text{Auxiliary trajectory dist. q}\\ \downarrow}}{q(\tau)} \left[\log p(O=1 \mid \tau) + \log \frac{p_\pi(\tau)}{q(\tau)}\right] d\tau \qquad \text{[Eq. 1]}$$

$$= \mathbb{E}_q\left[\textstyle\sum_t r_t/\alpha\right] - \mathrm{KL}(q(\tau) \parallel p_\pi(\tau)) = \underbrace{\mathcal{J}(q,\pi)}_{\textbf{ELBO}} \qquad \text{[Eq. 2]}$$

- EM can be used to alternately improve $\mathcal{J}$ w.r.t to (q,π)
  - E-step improves $\mathcal{J}$ w.r.t to q
  - M-step improves $\mathcal{J}$ w.r.t to π_θ supervised by reweighted q_i samples from E-step

MPO: Policy Improvement

- The infinite-horizon analogue with $q(\tau)$ factorization similar to p_π, i.e. $q(\tau) = p(s_0)\prod_{t>0} p(s_{t+1}|s_t, a_t)q(a_t|s_t)$ yields:

$$\mathcal{J}(q, \boldsymbol{\theta}) = \mathbb{E}_q\left[\sum_{t=0}^{\infty}\gamma^t\left[r_t - \alpha \mathrm{KL}\left(q(a \mid s_t) \parallel \pi(a \mid s_t, \boldsymbol{\theta})\right)\right]\right] + \log p(\boldsymbol{\theta}) \quad \text{[Eq. 3]}$$

Prior over policy parameters

- Associated Q-value function:

$$Q_\theta^q(s, a) = r_0 + \mathbb{E}_{q(\tau), s_0=s, a_0=a}\left[\sum_{t\geq 1}^{\infty}\gamma^t\left[r_t - \alpha \mathrm{KL}\left(q_t \parallel \pi_t\right)\right]\right] \quad \text{[Eq. 4]}$$

With $KL(q_t \,||\, \pi_t) = KL(q(a|s_t) \,||\, \pi(a|s_t, \theta))$

Policy Improvement: **E-Step**

E-step iteration i: perform partial maximization of $\mathcal{J}(q,\theta)$ *w.r.t* ***q*** given $\theta = \theta_i$
Setting $q = \pi_{\theta_i}$, estimate the *unregularized* (since $KL(q \,||\, \pi_i) = 0$) action-value function $Q^q_{\theta_i}(s,a)$

$$Q^q_{\theta_i}(s,a) = Q_{\theta_i}(s,a) = \mathbb{E}_{\tau_{\pi_i}, s_0=s, a_0=a}\left[\textstyle\sum_t^\infty \gamma^t r_t\right]$$ [Eq. 5]

Improve lower-bound $\mathcal{J}$ w.r.t to *q* with Q_{θ_i}
Expand $Q_{\theta_i}(s,a)$ using the Bellman operator: $T^{\pi,q} = \mathbb{E}_{q(a|s)}\left[r(s,a) - \alpha \mathrm{KL}\,(q \parallel \pi_i) + \gamma \mathbb{E}_{p(s'|s,a)}\left[V_{\theta_i}(s')\right]\right]$

Optimize 'one-step' KL-regularized (soft) objective

$$\begin{aligned} \max_q \bar{\mathcal{J}}_s(q,\theta_i) &= \max_q T^{\pi,q} Q_{\theta_i}(s,a) \\ &= \max_q \mathbb{E}_{\mu(s)}\left[\mathbb{E}_{q(\cdot|s)}\left[Q_{\theta_i}(s,a)\right] - \alpha \mathrm{KL}(q \parallel \pi_i)\right] \end{aligned}$$

Soft KL regularization

Final E-step Objective Constrained E-Step

$$\boxed{\begin{aligned} \max_q \bar{\mathcal{J}}_s(q,\theta_i) = \max_q \mathbb{E}_{\mu(s)}\left[\mathbb{E}_{q(a|s)}\left[Q_{\theta_i}(s,a)\right]\right] \\ \text{s.t. } \mathbb{E}_{\mu(s)}\left[\,\mathbf{KL}\big(q(a|s), \pi(a|s,\theta_i)\big)\right] < \epsilon \end{aligned}}$$

[Eq. 7]

N.B. Partial E-Step Maximizing and obtaining $q_i = \arg\max \bar{\mathcal{J}}(q,\theta_i)$ does not fully optimize $\mathcal{J}$ (since Q_{θ_i} is a constant w.r.t to q)

Policy Improvement: E-Step Variants

To solve this objective, we need choose a form for the variational policy $\boldsymbol{q(a|s)}$:

Final E-step Objective Constrained E-Step

$$\boxed{\begin{aligned} \max_q \bar{\mathcal{J}}_s(q, \theta_i) = \ \max_q \mathbb{E}_{\mu(s)}\left[\mathbb{E}_{q(a|s)}\left[Q_{\theta_i}(s,a)\right]\right] \\ \text{s.t. } \mathbb{E}_{\mu(s)}\left[\,\mathbf{KL}\left(q(a|s), \pi(a|s, \theta_i)\right)\right] < \epsilon \end{aligned}}$$

Option 1: Use a **parametric** variational distribution $q(a|s, \theta^q)$ with params θ^q
(explicit M-step becomes redundant) [*see appendix* (Alg. 3)]

Option 2: Use a **non-parametric** representation for $q(a|s)$ given by sample distribution over (a, s)
Fit a parametric policy in the M-step that generalize across state space [*see appendix* (Alg. 1, 2)]

$$q_i(a \mid s) \propto \pi(a \mid s, \boldsymbol{\theta}_i)\exp\left(\frac{Q_{\theta_i}(s,a)}{\eta^*}\right)$$

where η^* can be minimized by the convex dual function (*after which we can evaluate* $q_i(a|s)$):

$$g(\eta) = \eta\epsilon + \eta\int \mu(s)\log \int \pi(a \mid s, \boldsymbol{\theta}_i)\exp\left(\frac{Q_{\theta_i}(s,a)}{\eta}\right) dads$$

Policy Improvement: **M-Step**

With q_i from the E-step, optimize $\mathcal{J}$ w.r.t to θ to obtain updated policy $\boldsymbol{\theta_{i+1} = arg\,max\,\theta\ \mathcal{J}(q_i, \theta)}$:

$$\max_\theta \mathcal{J}(q_i, \theta) = \max_\theta \mathbb{E}_{\mu_q(s)}\left[\mathbb{E}_{q(a|s)}[\log \pi(a \mid s, \boldsymbol{\theta})]\right] + \log p(\boldsymbol{\theta})$$ [Eq. 10]

Essentially, a **supervised learning** step. With a Gaussian prior around the current policy $p(\theta)$

Then, the generalized M-Step is:

$$\max_\theta \mathcal{J}(q_i, \theta) = \max_\pi \mathbb{E}_{\mu_q(s)}\left[\mathbb{E}_{q(a|s)}[\log \pi(a \mid s, \boldsymbol{\theta})] - \lambda \mathrm{KL}\left(\pi(a \mid s, \boldsymbol{\theta}_i), \pi(a \mid s, \boldsymbol{\theta})\right)\right]$$ [Eq. 11]

Final M-step Objective

Corresponding hard-constrained form:

$$\boxed{\begin{array}{l} \max_\theta \mathcal{J}(q_i, \theta) = \max_\pi \mathbb{E}_{\mu_q(s)}\left[\mathbb{E}_{q(a|s)}[\log \pi(a \mid s, \boldsymbol{\theta})]\right] \\ \qquad\qquad \text{s.t. } \mathbb{E}_{\mu_q(s)}\left[\, \mathbf{KL}\left(\pi(a|s, \theta_i), \pi(a|s, \boldsymbol{\theta})\right)\right] < \epsilon \end{array}}$$ [Eq. 12]

Policy Evaluation

- Policy evaluation done in an off-policy setting, using policy evaluation operator from the Retrace algorithm (Munos et al. 2016) [5].
- Concretely, a neural network with parameters $\boldsymbol{\phi}$ represents Q-function $Q_{\theta_i}(s, a, \phi)$, and minimizes the squared loss:

$$\min_\phi L(\phi) = \min_\phi \mathbb{E}_{\mu_b(s), b(a|s)}\left[\left(Q_{\theta_i}(s_t, a_t, \phi) - Q_t^{\text{ret}}\right)^2\right]$$

$$Q_t^{\text{ret}} = Q_{\phi'}(s_t, a_t) + \sum_{j=t}^{\infty} \gamma^{j-t}\left(\prod_{k=t+1}^{j} c_k\right)\left[r(s_j, a_j) + \mathbb{E}_{\pi(a|s_{j+1})}[Q_{\phi'}(s_{j+1}, a)] - Q_{\phi'}(s_j, a_j)\right]$$

$$c_k = \min\left(1, \frac{\pi(a_k \mid s_k)}{b(a_k \mid s_k)}\right)$$

where, $Q_{\phi'}(s, a)$: output of target Q-network, $b(a|s)$: probabilities of an arbitrary behaviour policy (replay buffer)

Experiments

- Continuous Control tasks: DM Control suite [7], parkour environments [8]
  - Including the classical cart-pole and acrobot dynamical systems, 2D and Humanoid walking

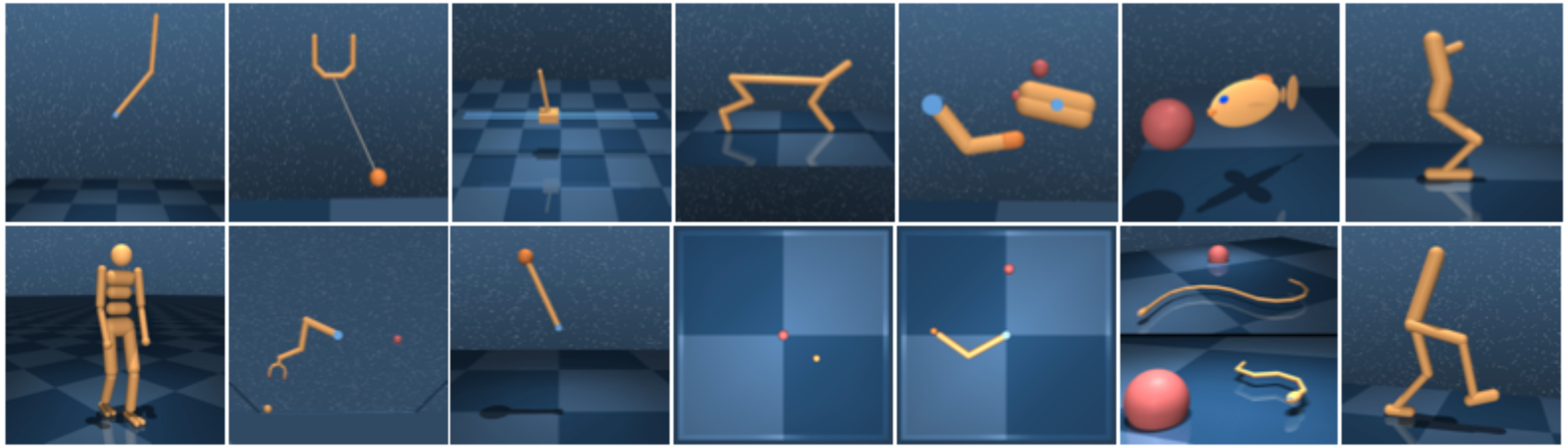
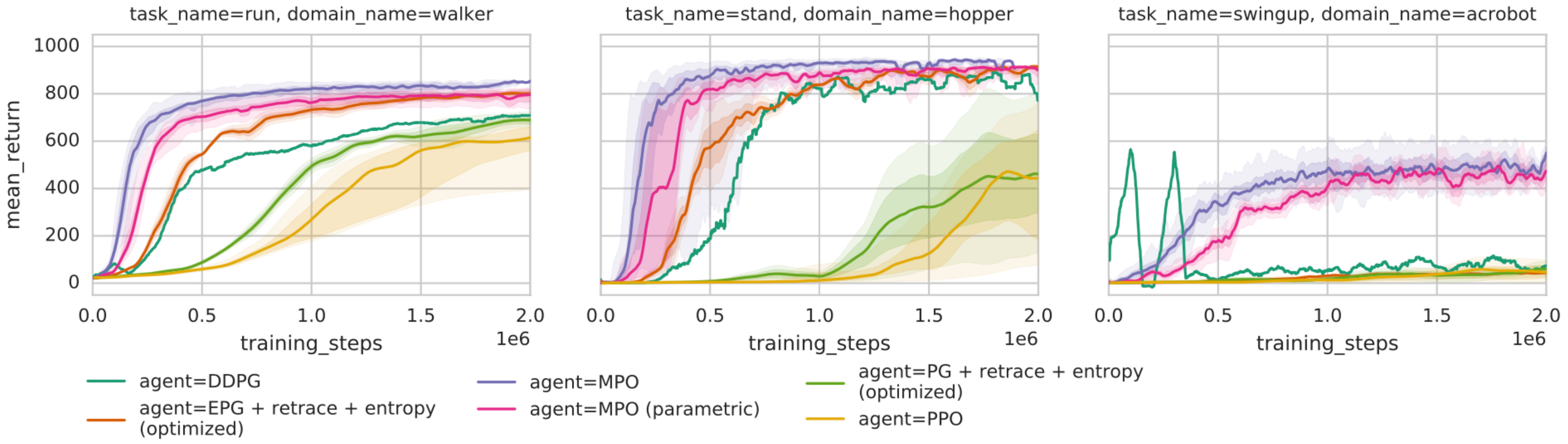


Figure 1: Control Suite domains used for benchmarking. *Top*: Acrobot, Ball-in-cup, Cart-pole, Cheetah, Finger, Fish, Hopper. *Bottom*: Humanoid, Manipulator, Pendulum, Point-mass, Reacher, Swimmers (6 and 15 links), Walker.

- Discrete Control tasks: ATARI environments [9]

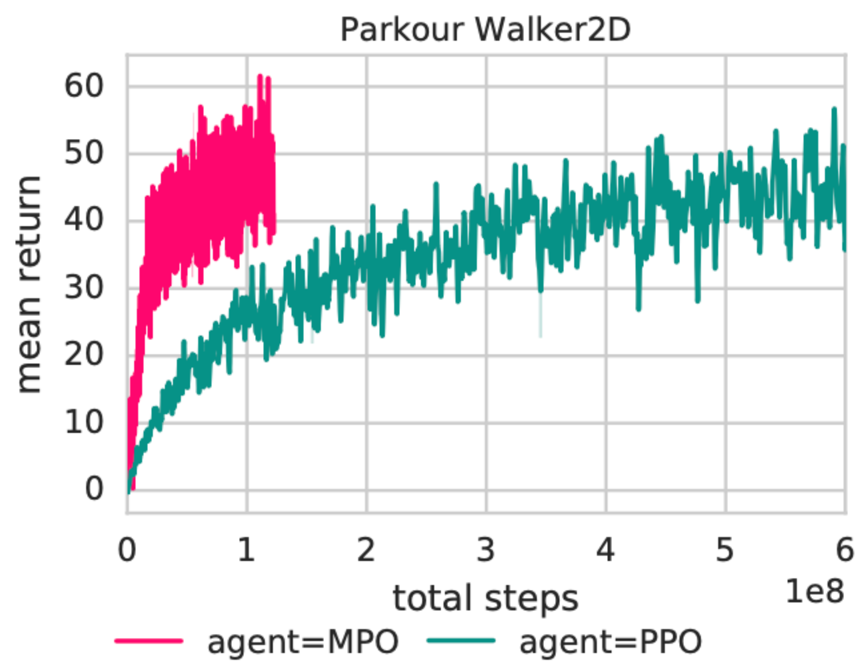
Experiments- Results

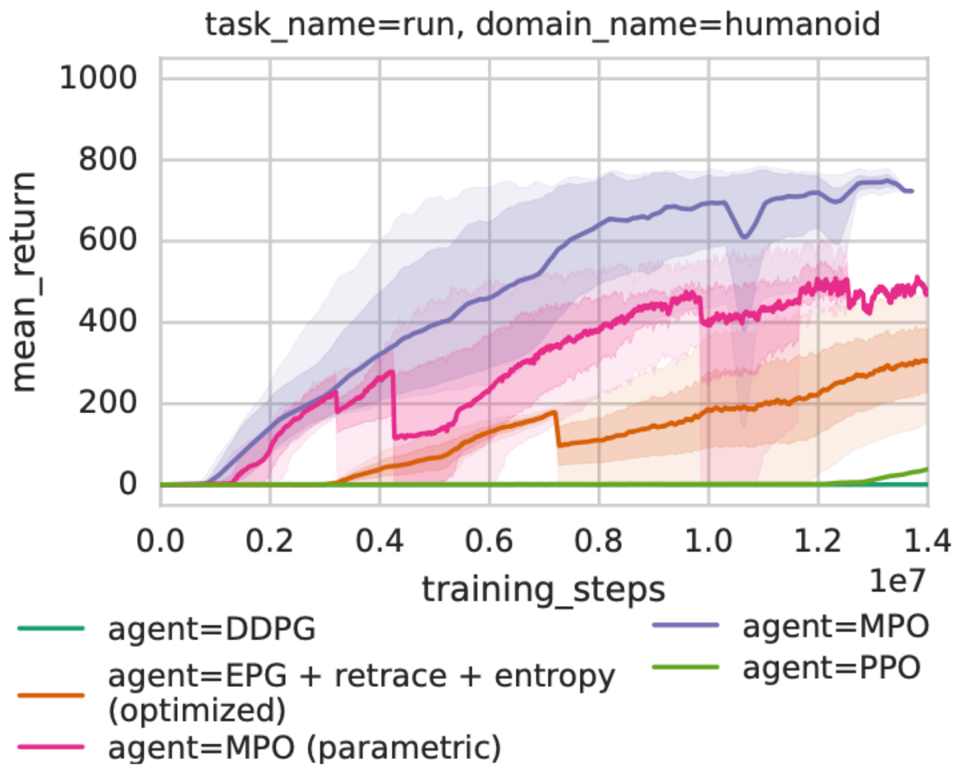
- Detailed evaluation on three (harder) tasks: Walker-2D, Hopper, Acrobot



Experiments – Results: High Dimensional Continuous Control

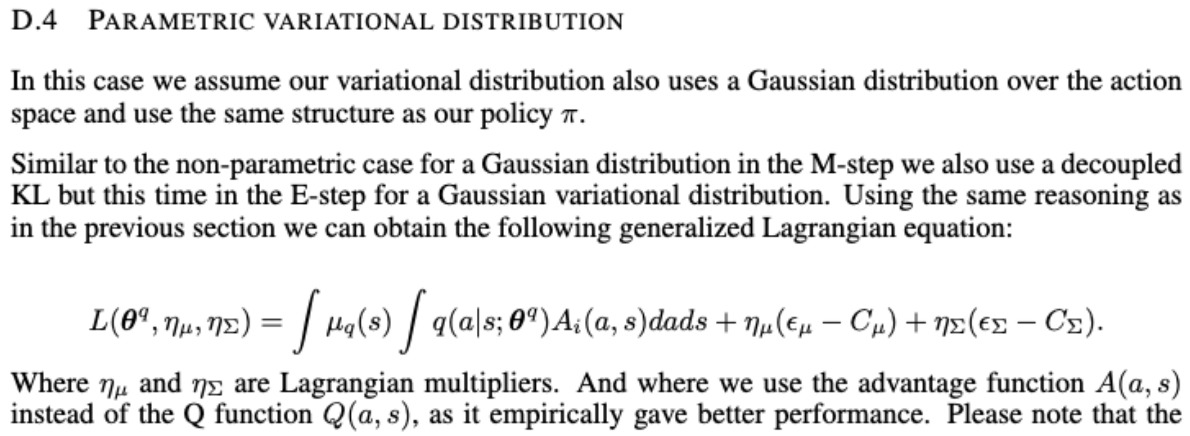
- High-dimensional Continuous Control - Parkour Walker 2D, Humanoid





Scope & Limitations

- Fairness of the baselines (sample-efficiency) comparison with on-policy methods (PPO, TRPO).
- Too many hyperparameters to tune (e.g. two KL constraints, for E and M step). Hard to stabilize training.
- Lot of implementation caveats (hacks) that deviate from the main motivation (MPO) optimization.



D.4 PARAMETRIC VARIATIONAL DISTRIBUTION

In this case we assume our variational distribution also uses a Gaussian distribution over the action space and use the same structure as our policy π.

Similar to the non-parametric case for a Gaussian distribution in the M-step we also use a decoupled KL but this time in the E-step for a Gaussian variational distribution. Using the same reasoning as in the previous section we can obtain the following generalized Lagrangian equation:

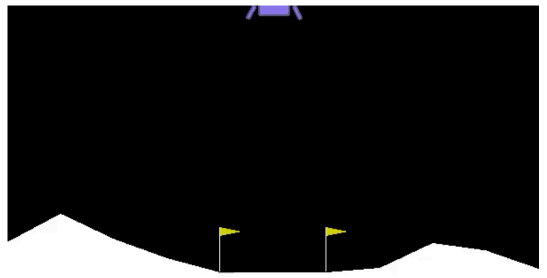
$$L(\boldsymbol{\theta}^q, \eta_\mu, \eta_\Sigma) = \int \mu_q(s) \int q(a|s; \boldsymbol{\theta}^q) A_i(a,s) da ds + \eta_\mu(\epsilon_\mu - C_\mu) + \eta_\Sigma(\epsilon_\Sigma - C_\Sigma).$$

Where η_μ and η_Σ are Lagrangian multipliers. And where we use the advantage function $A(a,s)$ instead of the Q function $Q(a,s)$, as it empirically gave better performance. Please note that the

Demo

- OpenAI Gym e.g.:
  - LunarLander-v2 (discrete) and LunarLanderContinuous-v2

Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

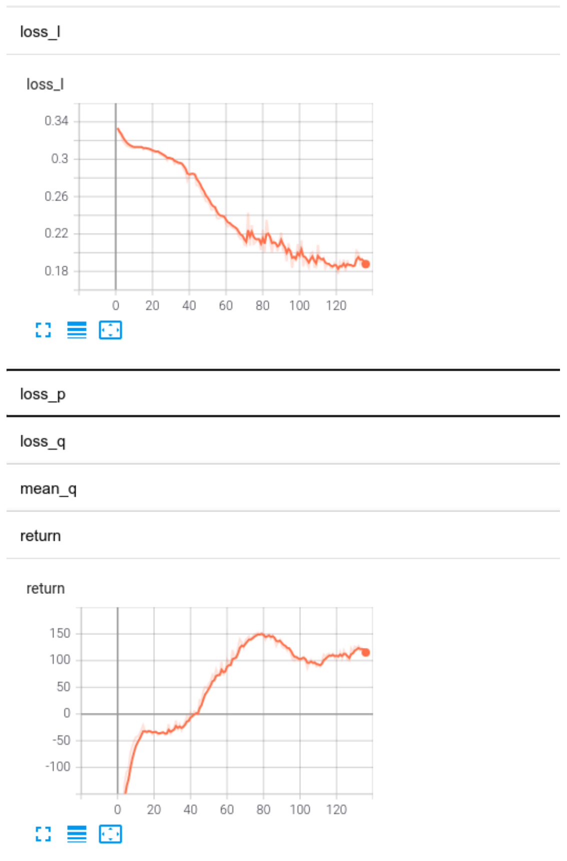


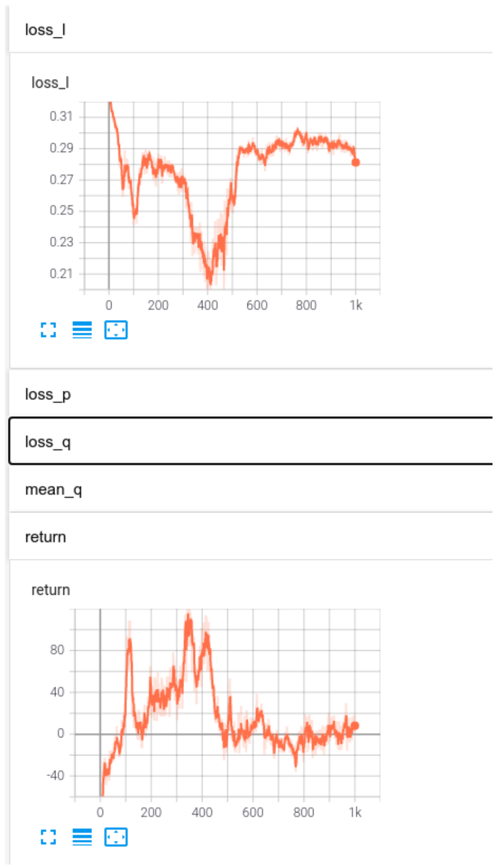
Source: https://gym.openai.com/envs/LunarLander-v2/

Insights

- Similarities with existing RL algorithms
  - TRPO, PPO
  - REPs

- Training can be treacherous





Summary

- The paper introduces a new off-policy RL algorithm: MPO
- The algorithm poses policy search as an inference problem and uses an alternating coordinate ascent style EM algorithm for policy improvement
- MPO empirically shows high data-efficiency, hyperparameter robustness and applicability to wide range of complex control problems.

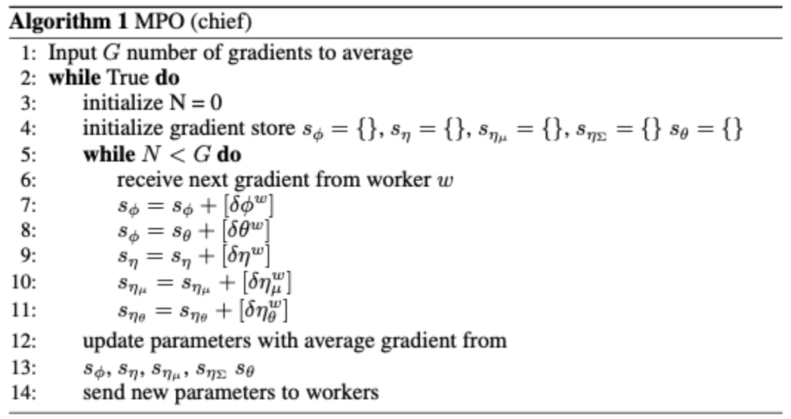
Questions?

Reference

1. Todorov, Emanuel, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control." 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2012.
2. Tassa, Yuval, et al. "Deepmind control suite." arXiv preprint arXiv:1801.00690 (2018).
3. J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, “Neural module networks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem.
4. J. A. Fodor *et al.*, “Connectionism and Cognitive Architecture Connectionism and Cognitive Architecture: 1 A Critical Analysis.”
5. Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.
6. Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
7. Munos, Rémi, et al. "Safe and efficient off-policy reinforcement learning." arXiv preprint arXiv:1606.02647 (2016).
8. Peters, Jan, Katharina Mulling, and Yasemin Altun. "Relative entropy policy search. (REPS)" Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 24. No. 1. 2010.
9. Tassa, Yuval, et al. "Deepmind control suite." arXiv preprint arXiv:1801.00690 (2018).
10. Heess, Nicolas, et al. "Emergence of locomotion behaviours in rich environments." arXiv preprint arXiv:1707.02286 (2017).
11. Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

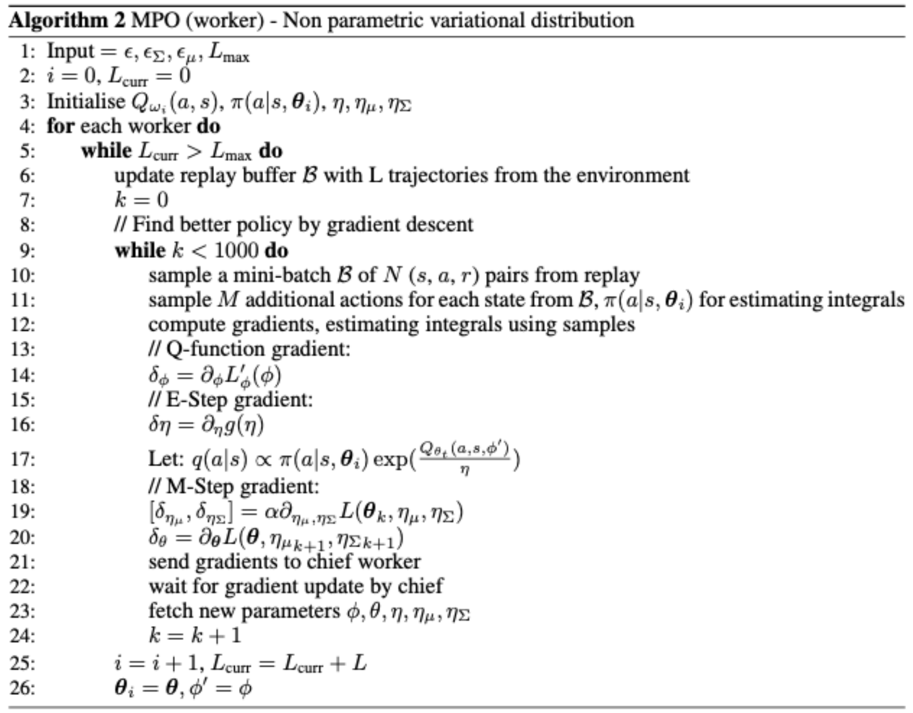
Appendix

Implementation Details



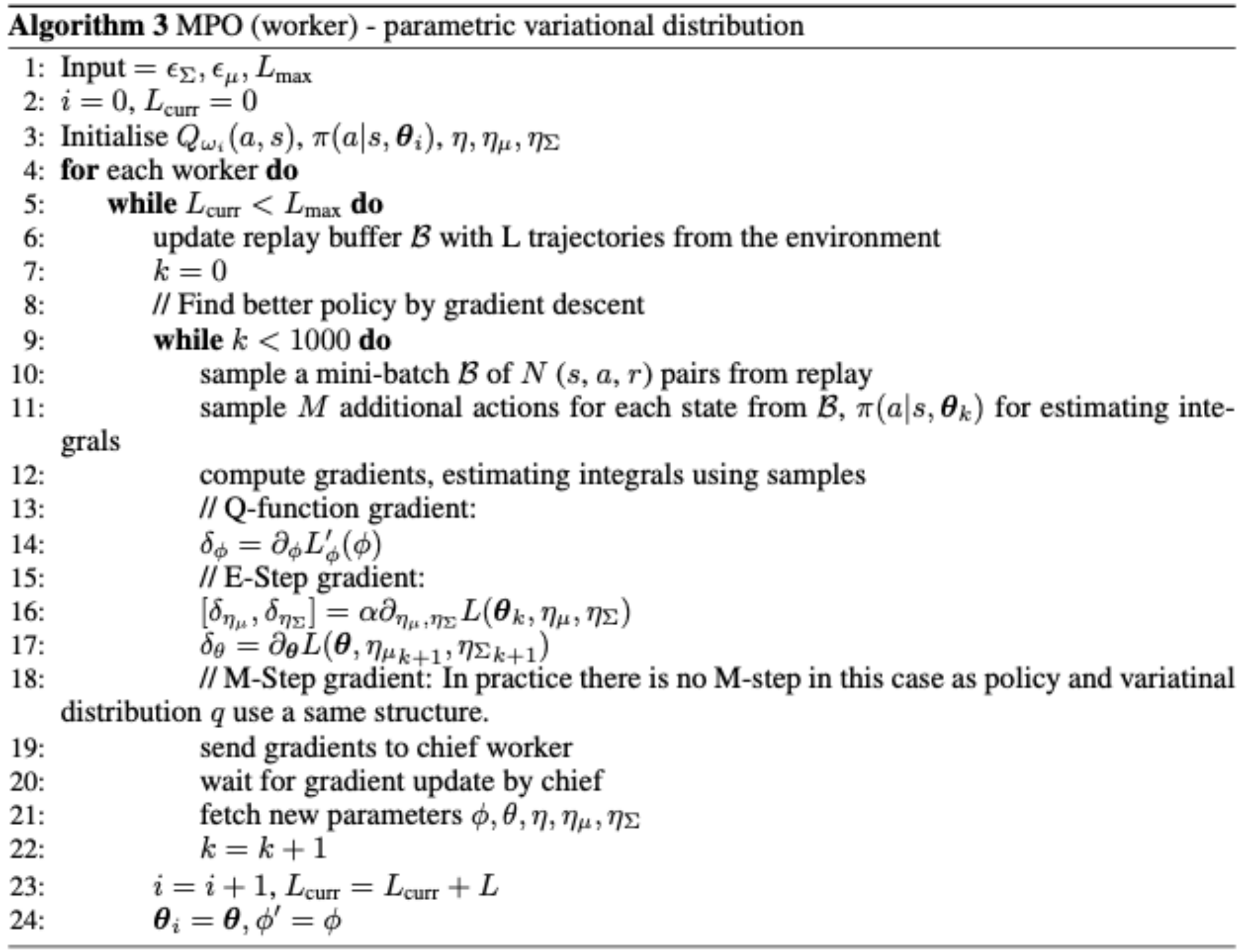
Algorithm 1 MPO (chief)

```
 1: Input $G$ number of gradients to average
 2: while True do
 3:     initialize N = 0
 4:     initialize gradient store $s_\phi = \{\}$, $s_\eta = \{\}$, $s_{\eta_\mu} = \{\}$, $s_{\eta_\Sigma} = \{\}$ $s_\theta = \{\}$
 5:     while $N < G$ do
 6:         receive next gradient from worker $w$
 7:         $s_\phi = s_\phi + [\delta\phi^w]$
 8:         $s_\phi = s_\theta + [\delta\theta^w]$
 9:         $s_\eta = s_\eta + [\delta\eta^w]$
10:         $s_{\eta_\mu} = s_{\eta_\mu} + [\delta\eta_\mu^w]$
11:         $s_{\eta_\theta} = s_{\eta_\theta} + [\delta\eta_\theta^w]$
12:     update parameters with average gradient from
13:     $s_\phi$, $s_\eta$, $s_{\eta_\mu}$, $s_{\eta_\Sigma}$ $s_\theta$
14:     send new parameters to workers
```



Algorithm 2 MPO (worker) - Non parametric variational distribution

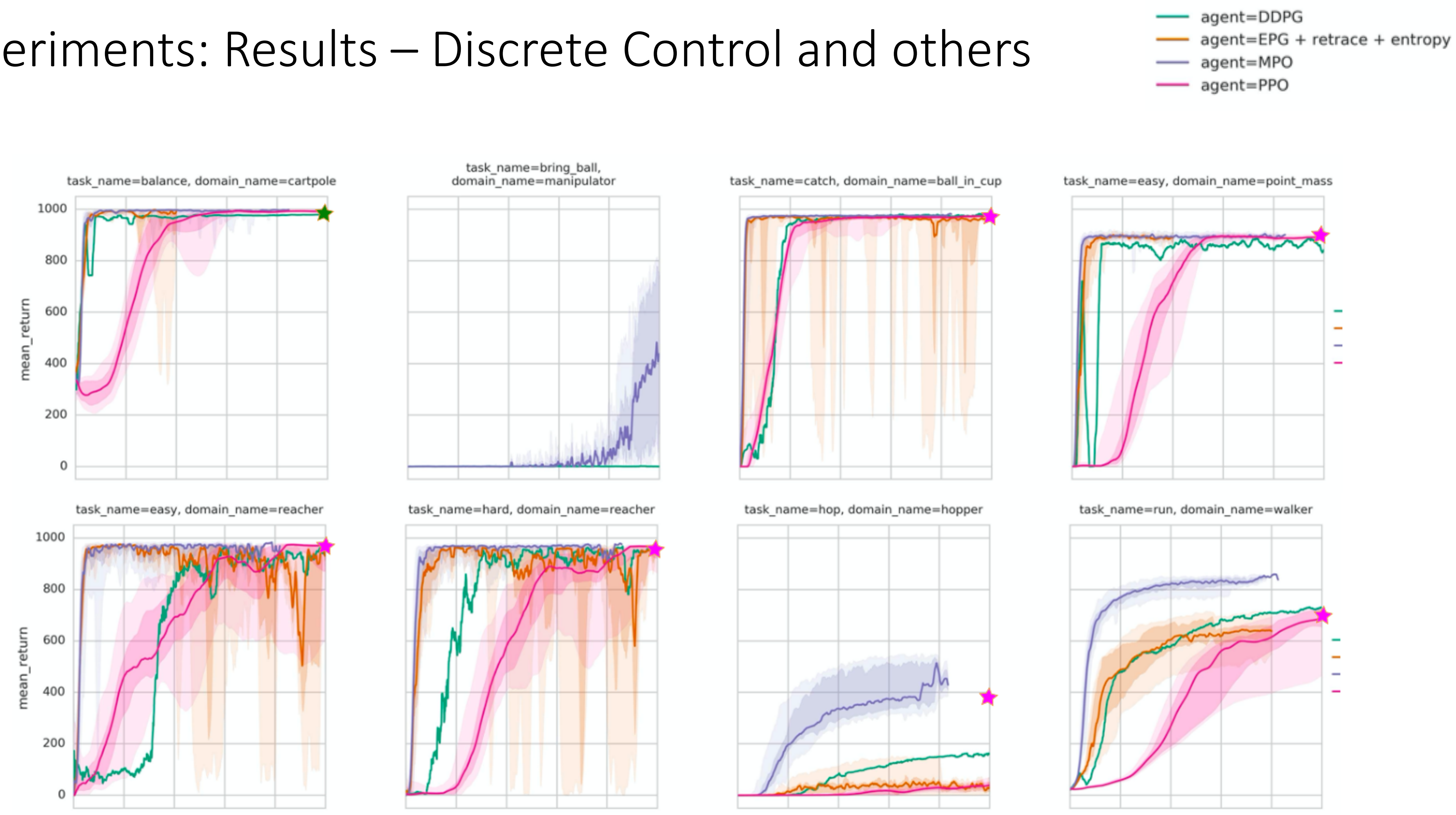
```
 1: Input $= \epsilon, \epsilon_\Sigma, \epsilon_\mu, L_{\text{max}}$
 2: $i = 0$, $L_{\text{curr}} = 0$
 3: Initialise $Q_{\omega_i}(a, s)$, $\pi(a|s, \boldsymbol{\theta}_i)$, $\eta, \eta_\mu, \eta_\Sigma$
 4: for each worker do
 5:     while $L_{\text{curr}} > L_{\text{max}}$ do
 6:         update replay buffer $\mathcal{B}$ with L trajectories from the environment
 7:         $k = 0$
 8:         // Find better policy by gradient descent
 9:         while $k < 1000$ do
10:             sample a mini-batch $\mathcal{B}$ of $N$ $(s, a, r)$ pairs from replay
11:             sample $M$ additional actions for each state from $\mathcal{B}$, $\pi(a|s, \boldsymbol{\theta}_i)$ for estimating integrals
12:             compute gradients, estimating integrals using samples
13:             // Q-function gradient:
14:             $\delta_\phi = \partial_\phi L'_\phi(\phi)$
15:             // E-Step gradient:
16:             $\delta\eta = \partial_\eta g(\eta)$
17:             Let: $q(a|s) \propto \pi(a|s, \boldsymbol{\theta}_i) \exp(\frac{Q_{\theta_t}(a,s,\phi')}{\eta})$
18:             // M-Step gradient:
19:             $[\delta_{\eta_\mu}, \delta_{\eta_\Sigma}] = \alpha \partial_{\eta_\mu, \eta_\Sigma} L(\boldsymbol{\theta}_k, \eta_\mu, \eta_\Sigma)$
20:             $\delta_\theta = \partial_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \eta_{\mu_{k+1}}, \eta_{\Sigma k+1})$
21:             send gradients to chief worker
22:             wait for gradient update by chief
23:             fetch new parameters $\phi, \theta, \eta, \eta_\mu, \eta_\Sigma$
24:             $k = k + 1$
25:         $i = i + 1$, $L_{\text{curr}} = L_{\text{curr}} + L$
26:         $\boldsymbol{\theta}_i = \boldsymbol{\theta}, \phi' = \phi$
```



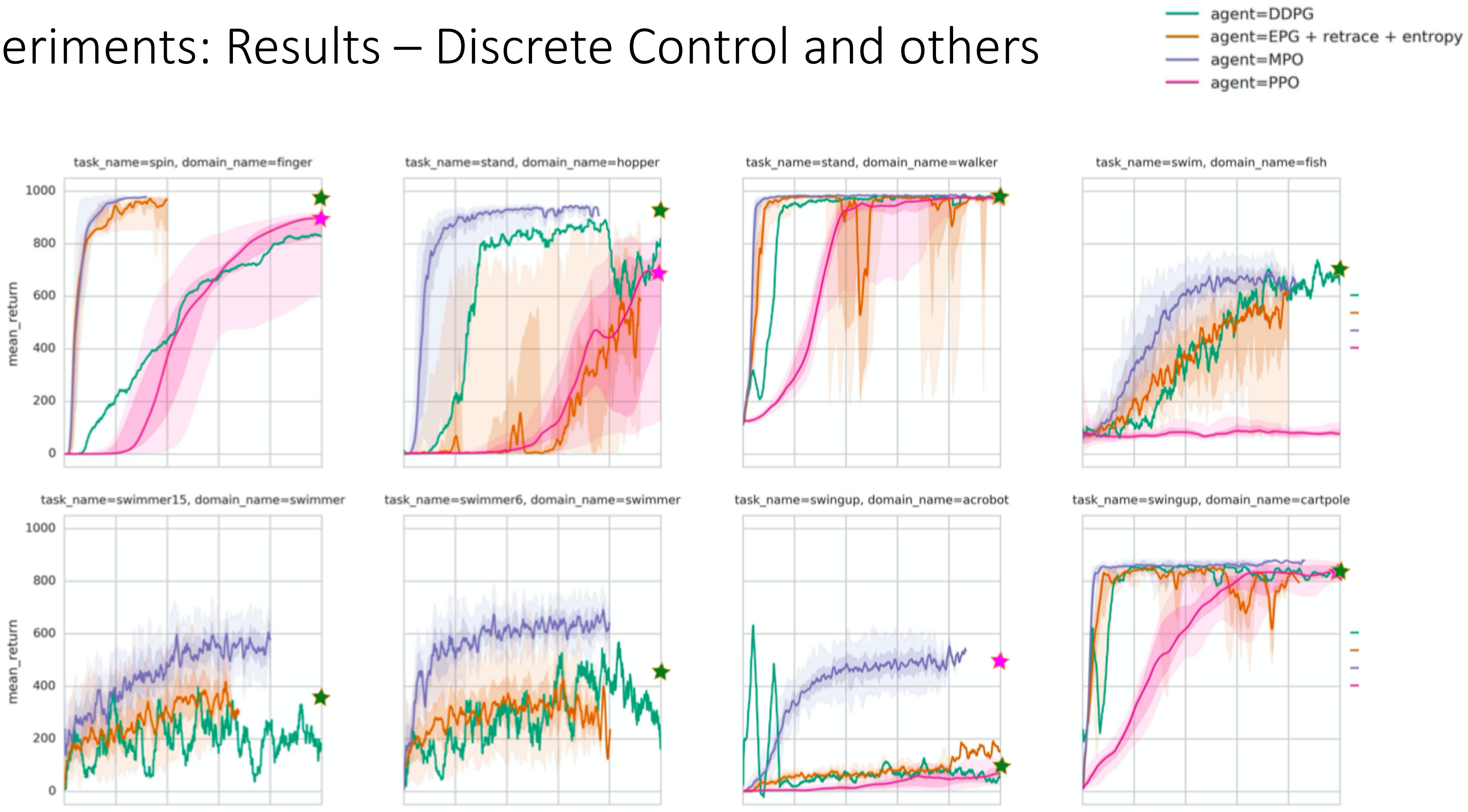
Algorithm 3 MPO (worker) - parametric variational distribution

```
 1: Input $= \epsilon_\Sigma, \epsilon_\mu, L_{\text{max}}$
 2: $i = 0$, $L_{\text{curr}} = 0$
 3: Initialise $Q_{\omega_i}(a, s)$, $\pi(a|s, \boldsymbol{\theta}_i)$, $\eta, \eta_\mu, \eta_\Sigma$
 4: for each worker do
 5:     while $L_{\text{curr}} < L_{\text{max}}$ do
 6:         update replay buffer $\mathcal{B}$ with L trajectories from the environment
 7:         $k = 0$
 8:         // Find better policy by gradient descent
 9:         while $k < 1000$ do
10:             sample a mini-batch $\mathcal{B}$ of $N$ $(s, a, r)$ pairs from replay
11:             sample $M$ additional actions for each state from $\mathcal{B}$, $\pi(a|s, \boldsymbol{\theta}_k)$ for estimating inte-
    grals
12:             compute gradients, estimating integrals using samples
13:             // Q-function gradient:
14:             $\delta_\phi = \partial_\phi L'_\phi(\phi)$
15:             // E-Step gradient:
16:             $[\delta_{\eta_\mu}, \delta_{\eta_\Sigma}] = \alpha \partial_{\eta_\mu, \eta_\Sigma} L(\boldsymbol{\theta}_k, \eta_\mu, \eta_\Sigma)$
17:             $\delta_\theta = \partial_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \eta_{\mu_{k+1}}, \eta_{\Sigma k+1})$
18:             // M-Step gradient: In practice there is no M-step in this case as policy and variatinal
    distribution $q$ use a same structure.
19:             send gradients to chief worker
20:             wait for gradient update by chief
21:             fetch new parameters $\phi, \theta, \eta, \eta_\mu, \eta_\Sigma$
22:             $k = k + 1$
23:         $i = i + 1$, $L_{\text{curr}} = L_{\text{curr}} + L$
24:         $\boldsymbol{\theta}_i = \boldsymbol{\theta}, \phi' = \phi$
```

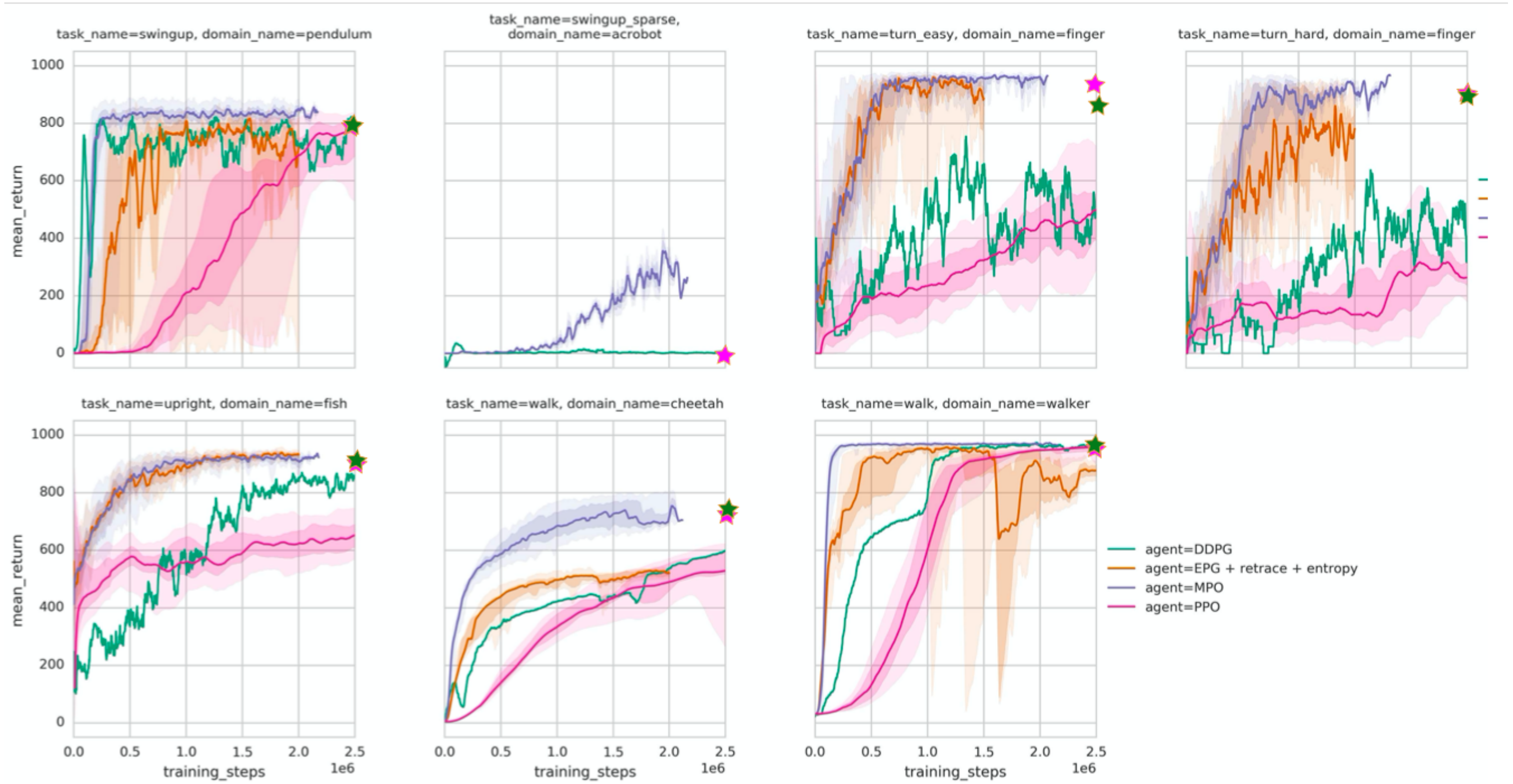
Experiments: Results – Discrete Control and others



Experiments: Results – Discrete Control and others



Experiments: Results – Discrete Control and others


task_name=swingup, domain_name=pendulum
task_name=swingup_sparse, domain_name=acrobot
task_name=turn_easy, domain_name=finger
task_name=turn_hard, domain_name=finger
task_name=upright, domain_name=fish
task_name=walk, domain_name=cheetah
task_name=walk, domain_name=walker
mean_return
training_steps
1e6
agent=DDPG
agent=EPG + retrace + entropy
agent=MPO
agent=PPO