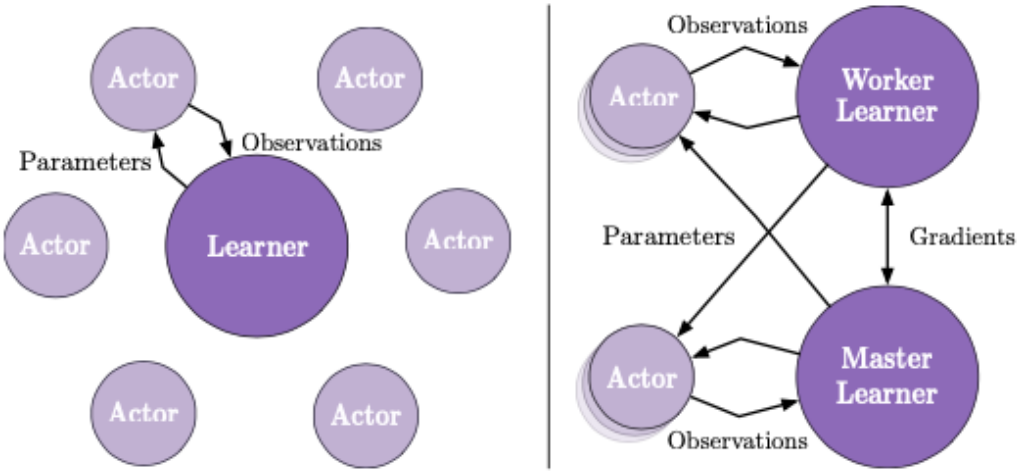
MPO

MAXIMUM A POSTERIORI POLICY OPTIMISATION

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos,
Nicolas Heess, Martin Riedmiller
DeepMind, London, UK
{aabdolmaleki,springenberg,tassa,munos,heess,riedmiller}@google.com

- backgrounds
  - handling off-policiness is important in the decoupled distributed actor-learner architecture such as IMPALA.



  - on-policy algorithms (e.g. TRPO or PPO) dominate in such a scalable learning framework but they suffer from poor sample efficiency.
- this paper proposes a new algorithm for RL called Maximum aposteriori Policy Optimisation (MPO) based on **coordinate ascent on a relative-entropy objective**.
  - MPO leverages the fast convergence properties of EM-style by alternating a non-parametric data-based E-step which re-weights state-action samples, with a supervised, parametric M-step using deep neural networks.
  - this makes MPO does *not require gradient of the Q-function to update the policy*

- an off-policy algorithm that benefits from the best properties of on-policy algorithms
  - scalability, robustness and hyperparameter insensitivity

MPO

▼ RL (maximize $\mathbb{E}_{\tau\sim\pi}[\sum_t \gamma^t r(s_t, a_t)]$) as probabilistic inference

- undiscounted reward formulation as inference problem
  - $p(\mathcal{O}=1|\tau) \propto \exp(\sum_t r_t/\alpha)$
  - $\mathcal{O}$ as *the event of obtaining maximum reward for given trajectory* τ
- evidence lowerbound (ELBO) of probability of policy π is optimal

$$\begin{aligned}\log p_\pi(\mathcal{O}=1) &= \log \int p_\pi(\tau) p(\mathcal{O}=1|\tau) d\tau \\ &\geq \int q(\tau)[\log p(\mathcal{O}=1|\tau) + \log \frac{p_\pi(\tau)}{q(\tau)}]d\tau \\ &= \mathbb{E}_q\left[\sum_t r_t/\alpha\right] - \mathrm{KL}\left(q(\tau)\|p_\pi(\tau)\right) = \mathcal{J}(q,\pi).\end{aligned} \tag{2}$$

  - optimizing (2) with respect to q can be seen as a KL regularized RL problem.
- optimizing $\mathcal{J}$ with EM algorithms which alternate between improving $\mathcal{J}$ with respect to q and π.

▼ Policy Improvement

- $p_\pi(\tau) = p(s_0)\prod_{t>=0} p(s_{t+1}|s_t, a_t)\pi(a_t, s_t)$
- $q(\tau) = p(s_0)\prod_{t>0} p(s_{t+1}|s_t, a_t)q(a_t, s_t)$
- then Eq.2 can be rerwitten as:

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

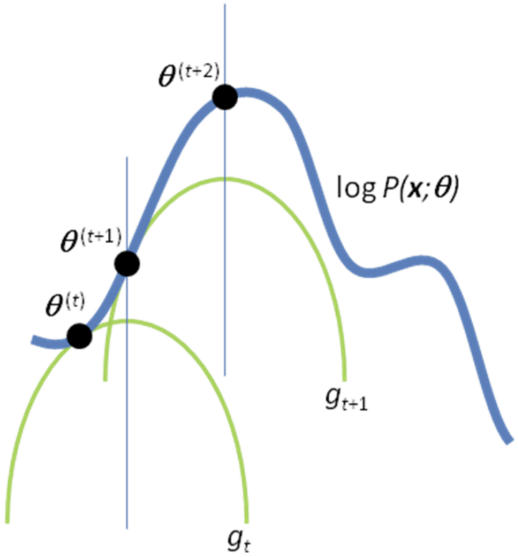
- Q-value function associated above equation can be defined as:
  $Q^q_\theta(s,a) = r_0 + \mathbb{E}_{q(\tau), s_0=s, a_0=a}\left[\sum_{t\geq 1}\gamma^t[r_t - \alpha\mathrm{KL}(q_t\|\pi_t)]\right].$

- $\mathrm{KL}(q_0 \| \pi_0)$ and $p(\theta)$ are not part of the Q function as they are not a function of the action.
- optimizing $\mathcal{J}$ w.r.t q is equivalent to solving an expected reward RL problem with augmented reward $\tilde{r} = r - \alpha \log \frac{q(a|s)}{\pi(a|s,\theta)}$

- π is the primary object of interest q is an auxiliary distribution that allows optimizing $\mathcal{J}$ via alternate coordinate ascent in q and π_θ.

  → analogous to the EM algorithm, E-step optimizes $\mathcal{J}$ w.r.t q while the M-step optimizes $\mathcal{J}$ w.r.t π_θ

  ▼ EM algorithm



Supplementary Figure 1 Convergence of the EM algorithm. Starting from initial parameters $\theta^{(t)}$, the E-step of the EM algorithm constructs a function g_t that lower-bounds the objective function $\log P(x;\theta)$. In the M-step, $\theta^{(t+1)}$ is computed as the maximum of g_t. In the next E-step, a new lower-bound g_{t+1} is constructed; maximization of g_{t+1} in the next M-step gives $\theta^{(t+2)}$, etc.

▼ E-step

Given $Q_{\boldsymbol{\theta}_i}$ we improve the lower bound $\mathcal{J}$ w.r.t. q by first expanding $Q_{\boldsymbol{\theta}_i}(s,a)$ via the regularized Bellman operator $T^{\pi,q} = \mathbb{E}_{q(a|s)}\Big[r(s,a) - \alpha \mathrm{KL}(q\|\pi_i) + \gamma \mathbb{E}_{p(s'|s,a)}[V_{\boldsymbol{\theta}_i}(s')]\Big]$, and optimize the "one-step" KL regularised objective

$$\begin{aligned}\max_q \bar{\mathcal{J}}_s(q,\theta_i) &= \max_q T^{\pi,q} Q_{\boldsymbol{\theta}_i}(s,a) \\ &= \max_q \mathbb{E}_{\mu(s)}\Big[\mathbb{E}_{q(\cdot|s)}[Q_{\boldsymbol{\theta}_i}(s,a)] - \alpha \mathrm{KL}(q\|\pi_i)\Big],\end{aligned} \tag{6}$$

since $V_{\boldsymbol{\theta}_i}(s) = \mathbb{E}_{q(a|s)}[Q_{\boldsymbol{\theta}_i}(s,a)]$ and thus $Q_{\boldsymbol{\theta}_i}(s,a) = r(s,a) + \gamma V_{\boldsymbol{\theta}_i}(s)$.

Maximizing Equation (6), thus obtaining $q_i = \arg\max \bar{\mathcal{J}}(q,\theta_i)$, does not fully optimize $\mathcal{J}$ since we treat Q_{θ_i} as constant with respect to q. An intuitive interpretation q_i is that it chooses the soft-optimal action for one step and then resorts to executing policy π. In the language of the EM algorithm this optimization implements a partial E-step. In practice we also choose μ_q to be the stationary distribution as given through samples from the replay buffer.

▼ Constrained E-step

The reward and the KL terms are on an arbitray relative scale. This can make it difficult to choose α. We therefore replace the soft KL regularization with a hard constraint with parameter ϵ, i.e,

$$\begin{aligned}&\max_q \mathbb{E}_{\mu(s)}\Big[\mathbb{E}_{q(a|s)}\Big[Q_{\theta_i}(s,a)\Big]\Big] \\ &s.t. \mathbb{E}_{\mu(s)}\Big[\mathrm{KL}(q(a|s), \pi(a|s,\boldsymbol{\theta}_i))\Big] < \epsilon.\end{aligned} \tag{7}$$

- To implement (7), it is required to choose a form for the variational policy $q(a|s)$. Two options arise:
  - parameteric distribution $q(a|s,\theta^q)$ → an algorithm (TRPO or PPO) and M-step becomes unnecessary
  - non-parametric distribution $q(a|s)$
    - we can obtain the optimal sample based q distribution over actions for each state (see appendix for derivation)

$$q_i(a|s) \propto \pi(a|s,\boldsymbol{\theta}_i) \exp\Big(\frac{Q_{\theta_i}(s,a)}{\eta^*}\Big), \tag{8}$$

where we can obtain η^* by minimising the following convex dual function,

$$g(\eta) = \eta\epsilon + \eta \int \mu(s) \log \int \pi(a|s,\boldsymbol{\theta}_i) \exp\Big(\frac{Q_{\theta_i}(s,a)}{\eta}\Big) da ds, \tag{9}$$

    - It allows us to estimate the integral over actions with multiple samples without additional environment interaction (optimize for $q(a|s)$ instead of $q(a,s)$). This greatly reduces the variance of the estimate and allows for fully off-policy learning at the cost of performing only a partial optimization of $\mathcal{J}$

▼ M-step

- After dropping unrelevant terms about parameter θ of π,

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

  - corresponds to a weighted maximum a-posteriroi estimation (MAP) problem where samples are weighted by the variational distribution from the E-step.

- with a Gaussian prior $p(\theta) \approx \mathcal{N}(\mu = \theta_i, \Sigma = \frac{F_{\theta_i}}{\lambda})$ where F_{θ_i} is the empirical Fisher information matrix, optimization program becomes

$$J(\boldsymbol{s}, \pi) = \mathbb{E}_{A \sim \pi(\cdot|s)} [\hat{q}_{\pi_{\text{old}}}(\boldsymbol{s}, A)] - \lambda \text{KL}(\pi, \pi_{\text{old}})$$

- constrained version

$$\max_{\pi} \mathbb{E}_{\mu_q(s)} \Big[\mathbb{E}_{q(a|s)} \Big[\log \pi(a|s, \boldsymbol{\theta}) \Big] \Big]$$
$$s.t.\ \mathbb{E}_{\mu_q(s)} \Big[\mathbf{KL}(\pi(a|s, \boldsymbol{\theta}_i), \pi(a|s, \boldsymbol{\theta})) \Big] < \epsilon.$$

▼ Policy Evaluation

- Used of the policy evaluation operator from the *Retrace* algorithm Munos et al. (2016)

$$\min_{\phi} L(\phi) = \min_{\phi} \mathbb{E}_{\mu_b(s), b(a|s)} \Big[\big(Q_{\theta_i}(s_t, a_t, \phi) - Q_t^{\text{ret}} \big)^2 \Big], \text{with}$$
$$Q_t^{\text{ret}} = Q_{\phi'}(s_t, a_t) + \sum_{j=t}^{\infty} \gamma^{j-t} \Big(\prod_{k=t+1}^{j} c_k \Big) \Big[r(s_j, a_j) + \mathbb{E}_{\pi(a|s_{j+1})} [Q_{\phi'}(s_{j+1}, a)] - Q_{\phi'}(s_j, a_j) \Big],$$
$$c_k = \min \Big(1, \frac{\pi(a_k|s_k)}{b(a_k|s_k)} \Big),$$
(13)

where $Q_{\phi'}(s, a)$ denotes the output of a target Q-network, with parameters ϕ', that we copy from the current parameters ϕ after each M-step. We truncate the infinite sum after N steps by bootstrapping with $Q_{\phi'}$ (rather than considering a λ return). Additionally, $b(a|s)$ denotes the probabilities of an arbitrary behaviour policy. In our case we use an experience replay buffer and hence b is given by the action probabilities stored in the buffer; which correspond to the action probabilities at the time of action selection.

Algorithm in code level (in descrete action spaces)

- referenced from https://github.com/daisatojp/mpo/blob/master/mpo/mpo.py

▼ E-step (finding the action weights in Eq.8)

- finding $Q_{\theta_i}(s, a)$

```
b_p = self.target_actor.forward(state_batch)  # (K, da)
b = Categorical(probs=b_p)  # (K,)
b_prob = b.expand((da, K)).log_prob(actions).exp()  # (da, K)
expanded_actions = self.A_eye[None, ...].expand(K, -1, -1)  # (K, da, da)
expanded_states = state_batch.reshape(K, 1, ds).expand((K, da, ds))  # (K, da, ds)
target_q = (
    self.target_critic.forward(
        expanded_states.reshape(-1, ds),  # (K * da, ds)
        expanded_actions.reshape(-1, da)  # (K * da, da)
    ).reshape(K, da)  # (K, da)
).transpose(0, 1)  # (da, K)
```

- finding η

```
def dual(η):
  max_q = np.max(target_q_np, 1)
  return η * self.ε_dual + np.mean(max_q) \
      + η * np.mean(np.log(np.sum(
          b_prob_np * np.exp((target_q_np - max_q[:, None]) / η), axis=1)))
```

- then

```
bounds = [(1e-6, None)]
res = minimize(dual, np.array([self.η]), method='SLSQP', bounds=bounds)
η = res.x[0]
qij = torch.softmax(target_q / η, dim=0)  # (N, K) or (da, K)
```

▼ M-step

```
π_p = self.actor.forward(state_batch)  # (K, da)
π = Categorical(probs=π_p)  # (K,)
loss_p = torch.mean(qij * π.expand((da, K)).log_prob(actions))
mean_loss_p.append((-loss_p).item())

kl = categorical_kl(p1=π_p, p2=b_p)
max_kl.append(kl.item())

self.α -= self.α_scale * (self.ε_kl - kl).detach().item()
self.α = np.clip(self.α, 0.0, self.α_max)
```

```
actor_optimizer.zero_grad()

loss_l = -(loss_p + self.α * (self.ε_kl - kl))
mean_loss_l.append(loss_l.item())
loss_l.backward()
clip_grad_norm_(self.actor.parameters(), 0.1)
actor_optimizer.step()
```

Experiments (Control Suite)

Figure 2: Ablation study of the MPO algorithm and comparison to common baselines from the literature on three domains from the control suite. We plot the median performance over 10 experiments with different random seeds.

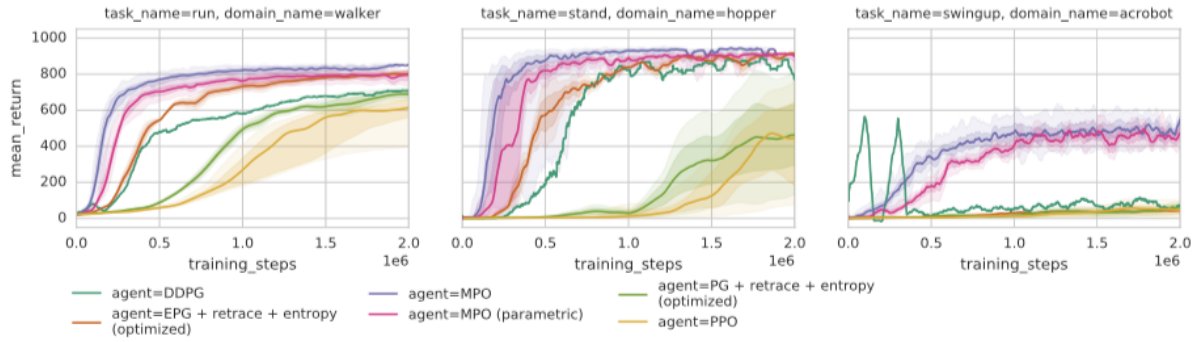


Figure 3: MPO on high-dimensional control problems (Parkour Walker2D and Humanoid walking from control suite).