Model Predictive Control and Reinforcement Learning
– Lecture 7.1: An MPC prior for SAC –

Jasper Hoffmann

University of Freiburg

Fall School on Model Predictive Control and Reinforcement Learning
Freiburg, 6-10 October 2025

universität freiburg

NTNU | Norwegian University of Science and Technology

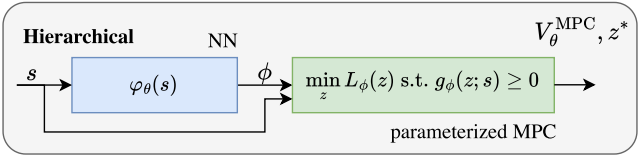
Online RL in the Real World

Dream: Deployed agents learn in the real world

Reality: RL unsafe, sample-inefficient, local maxima, hard exploration, spurious correlations

Idea: MPC engineering prior to guide learning and ensure safety

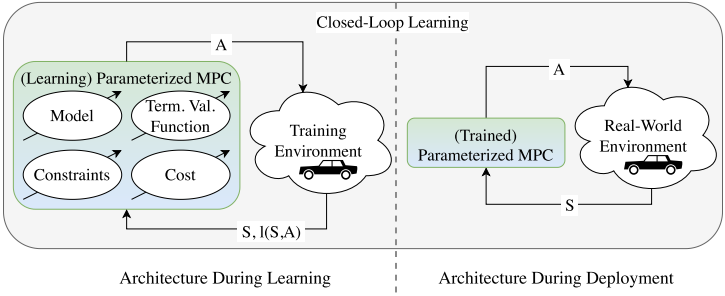
Ingredient 1: Hierarchical Structure



Motivation:

- Hierarchical structure allows usage of MPC as a safety filter
- Using MPC as a low level controller allows us to potentially simplify the RL problem
- Nonlinearity of $\varphi_\theta(s)$ does not affect optimization structure, OCP can be efficiently solved

Ingredient 2: Closed-loop learning



Motivation:

- Parameterize the MPC such that we optimize closed-loop performance
- Focus on what **works best**, not what is most **accurate**. Internal models/costs may become misaligned

Caveat:

- We could also combine this with aligned learning $\implies$ interpretability, safety etc.
- To keep it simple we focus on closed-loop learning

Example: SAC without differentiable MPC

What has been done so far?

Started from a MPCRL school project:

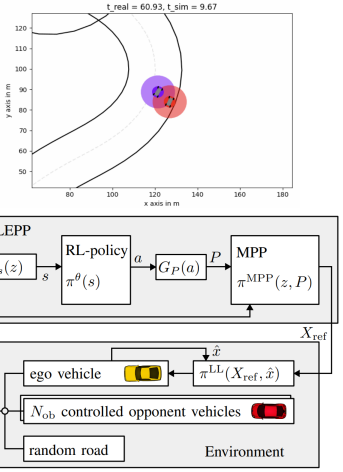
- Interactive car-racing
- MPC (acados) takes care of obstacle avoidance and staying on track in curves
- RL strategically positions the car in different scenarios:
  - Overtaking a slower car
  - Blocking a faster car
  - Overtaking and blocking

No differentiation through MPC scheme?

2023 European Control Conference (ECC)
June 13-16, 2023. Bucharest, Romania

A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing

Rudolf Reiter[1], Jasper Hoffmann[2], Joschka Boedecker[2] and Moritz Diehl[1,3]



Example: PPO with differentiable MPC

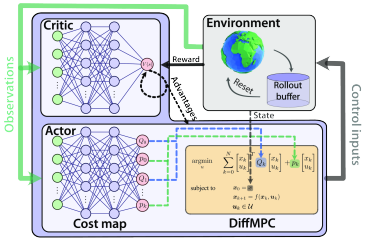
Differentiable MPC within PPO

- Drone needs to navigate through a race track marked by gates
- Parameterize quadratic cost function
- It uses a differentiable MPC formulation in the actor
- One of the first works to integrate differentiable MPC into RL

However, PPO is an on-policy actor–critic method that only retains transition data briefly, limiting the sample efficiency.

Actor-Critic Model Predictive Control

Angel Romero, Yunlong Song, Davide Scaramuzza



Our Attempt

`leap-c`

Modular Code:

- Provide a differentiable MPC layer for PyTorch
- Use a state-of-the-art solver like `acados` to achieve sufficient speed

Benchmarking:

- Learning control community rarely benchmarks across multiple problems
- We provide diverse environments to enable fair comparison

`MPC Prior for SAC`

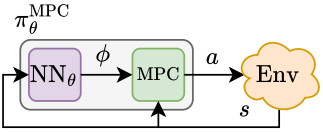
Integrate MPC into SAC:

- Use a state-of-the-art off-policy method like SAC for sample efficiency
- **Investigate questions:** Differentiable MPC? How to explore? ...

Outline of the lecture

Outline of the lecture

Hierarchical policy:



When building our hierarchical policy, we face three fundamental design choices:

1. **OCP Formulation:** How do we define the dynamics, cost, and constraints? This choice imposes the inductive bias and determines the trade-off between safety and optimality.
2. **Action vs. Parameter Critic:** Should the critic learn a value function over actions or over MPC parameters? This impacts the training procedure and computational graph.
3. **Exploration Strategy:** Where in the hierarchical policy do we inject noise for exploration — on the parameters or on the actions?

We will now discuss each of the points in detail.

Reminder: Parameterized OCP

Parameterized OCP with parameters ϕ:

OCP:

$$
\begin{aligned}
\min_{z} \quad & L_\phi(z) \\
\text{s.t.} \quad & x_0 = s, \\
& x_{k+1} = f_\phi^{\text{MPC}}\big(x_k, u_k\big),\ 0 \le k < N, \\
& 0 \le h_\phi^{\text{MPC}}\big(x_k, u_k\big),\ 0 \le k < N, \\
& 0 \le \tilde{h}_\phi^{\text{MPC}}\big(x_N\big),
\end{aligned}
$$

Objective:

$$
L_\phi(z) := \bar{V}_\phi^{\text{MPC}}(x_N) + \sum_{k=0}^{N-1} l_\phi^{\text{MPC}}\big(x_k, u_k\big).
$$

Short notation:

$$
\min_{z} L_\phi(z) \quad \text{s.t.} \quad g_\phi(z; s) \ge 0.
$$

Parameters ϕ: Can affect objective L_ϕ, dynamics f_ϕ^{MPC}, and constraints h_ϕ^{MPC}.
Parameter Space: We denote the corresponding parameter space with Φ.

Control Solution Map: Let $a = u_0^\star(s, \phi)$ denote the first control of the opt. solution $z^\star(s, \phi)$.

The impact of the OCP formulation

Realizable Action Set: The choice of dynamics, costs, and constraints determines the set of possible actions $\mathcal{A}_{\mathrm{MPC}}(s)$:

$$\mathcal{A}_{\mathrm{MPC}}(s) = \{u_0^\star(s, \phi) \mid \phi \in \Phi\} \subseteq \mathcal{A},$$

- A more expressive formulation (e.g., using a general-purpose solver like acados) expands this set, reducing the risk of sub-optimality.

Safety vs. Optimality Trade-off:

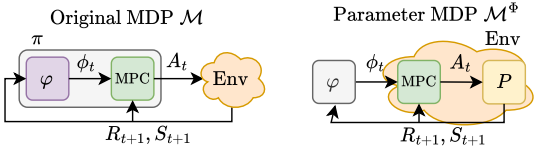
- Hard constraints in the MPC can guarantee safety by ensuring all actions are feasible.
- Yet, the policy becomes conservative if the truly optimal action lies outside the feasible set.

Spurious Local Minima: The mapping from parameters ϕ to actions $a = u_0^\star(s, \phi)$ can introduce local minima in the policy learning landscape, even if the critic is perfect.

- We showed that given a condition to avoid this: the Jacobian of the solution map $u_0^\star$ must have full row rank.

Parameter Critic vs Action Critic: Parameter MDP

Idea: The MPC can be seen as part of the environment!



- Learn *parameter policy* φ over parameters $\phi \in \Phi$ instead of policy π over actions $a \in \mathcal{A}$.
- The parameter space Φ, then becomes the new action space $\mathcal{A}$.
- The transition function then becomes $P^\Phi(\cdot \mid s, \phi) = P(\cdot \mid s, u_0^\star(s, \phi))$

Relationships & New objects:

- The parameter policy φ, induces an action policy $\pi(s) = u_0^\star(s, \varphi(s))$. The action policy is a policy in the original MDP $\mathcal{M}$.
- φ has corresponding value functions V^φ and Q^φ
- Assuming that the solution map $u_0^\star$ is deterministic, we have for all $\phi \in \Phi$ that

$$Q^\varphi(s, \phi) = Q^\pi(s, u_0^\star(s, \phi))$$

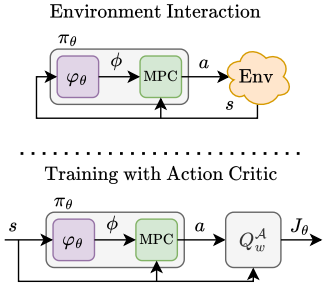
Parameter Critic vs Action Critic: Comparison

Action Critic: $Q_w^{\mathcal{A}}(s,a) \approx Q^{\pi}(s,a)$

- Provides feedback on action $a = u_0^\star(s, \phi)$.
- **Actor objective:**

$$J(\theta) = \mathbb{E}_{S\sim D}\big[Q_w^{\mathcal{A}}\big(S, u_0^\star(S, \varphi_\theta(S))\big)\big]$$

- **Requires differentiable MPC**:
  Gradients must flow back through the MPC scheme to the parameter network.



Parameter Critic vs Action Critic: Comparison

Parameter Critic: $Q_w^\Phi(s,\phi) \approx Q^\varphi(s,\phi)$

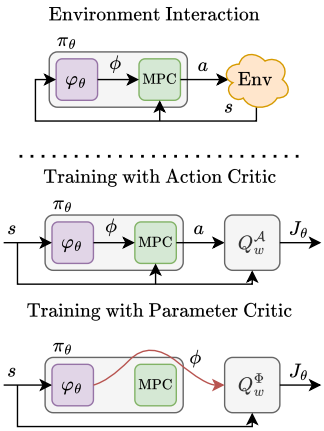
- Provides feedback on parameters ϕ.
- **Actor objective:**

$$J(\theta) = \mathbb{E}_{S\sim D}\big[Q_w^\Phi(S, \varphi_\theta(S))\big]$$

- MPC is treated as part of the environment, **bypassed during training updates**.

Summary:

- Parameter critic avoids MPC during training, enabling potentially faster training (can enable a simple pure GPU training loop).
- Action critic incorporates the sensitivities of the MPC scheme, potentially leading to more informed updates.



Exploration Strategies: Where to inject the Noise?

Exploration via Action Noise

Add noise $\xi_\theta^{\mathcal{A}}$ **directly to the final action** A:

$$A = u_0^\star\big(s,\, \phi_\theta(s)\big) + \xi_\theta^{\mathcal{A}}(s)$$

Implications:

- Can result in **infeasible or unsafe actions**.
- Noise might push the action outside MPC constraints.
- However, we have an explicit exploration in the action space.

Exploration via Parameter Noise

Noise ξ_θ^{Φ} is added to the **MPC parameters** ϕ **before** solving MPC:

$$A = u_0^\star\big(s,\, \phi_\theta(s) + \xi_\theta^{\Phi}(s)\big)$$

Implications:

- The MPC layer acts as a **filter**, ensuring actions remain feasible.
- Enables **safe exploration** while respecting system constraints.
- However, exploration is now **indirect** and transformed by the MPC.

Outline of the lecture

Hierachical Actor Architecture:

- A neural network models a Gaussian distribution over parameters:

$$\varphi_\theta(\cdot \mid s) = \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$$

with mean $\mu_\theta(s)$ and a covariance $\Sigma_\theta(s)$. We can generate samples using the reparameterization trick:

$$y_\theta(s, \xi) = \mu_\theta(s) + {\Sigma_\theta}^{1/2}\, \xi,$$

where $\xi \sim \mathcal{N}(0, I)$ is standard normal noise. Note, it holds that $y_\theta(s, \xi) \sim \varphi_\theta(\cdot \mid s)$.

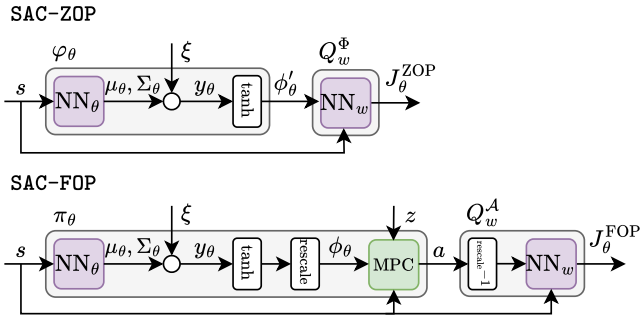
- We use a squashing function to ensure the parameters remain within bounds:

$$\phi_\theta(s, \xi) = \mathrm{rescale}\big[\tanh\big(y_\theta(s, \xi)\big)\big]$$

- The final policy is then $\pi_\theta(\cdot \mid s) \sim u_0^\star\big(s,\, \phi_\theta(s, \xi)\big)$.

We propose two variants that train π_θ.

Two Flavours: SAC-ZOP and SAC-FOP



SAC-ZOP (Zero-Order, Parameter Noise)

- Uses a **parameter critic** $Q_w^\Phi(s, \phi)$.
- **Avoids** differentiating through the MPC layer during training.
- Zero-order with respect to the MPC.

SAC-FOP (First-Order, Parameter Noise)

- Uses an **action critic** $Q_w^\mathcal{A}(s, a)$.
- **Requires** differentiating through the MPC layer for the actor update.
- Leverages first-order gradient information from the MPC solution map.

SAC-ZOP: Objectives

Note: We measure the entropy of the parameter policy φ_θ, not of the action policy π_θ:

$$\mathcal{H}(\varphi_\theta) = \mathbb{E}_{S\sim\mathcal{D},\phi\sim\varphi_\theta(\cdot|S)}\left[-\log\varphi_\theta(\phi \mid S)\right].$$

SAC-ZOP (Parameter Critic)

- **Actor Objective:** The actor maximizes

$$J^{\mathrm{ZOP}}(\theta) \coloneqq \mathbb{E}_{\substack{S\sim\mathcal{D}\\ \phi\sim\varphi_\theta(\cdot|S)}}\Big[Q_w^\Phi(S,\phi) - \alpha\log\left(\varphi_\theta(\phi \mid S)\right)\Big].$$

- **Critic Loss:** The critic minimizes the soft Bellman error

$$\mathcal{L}^{\mathrm{ZOP}}(w) = \frac{1}{2}\mathbb{E}_{(S,\phi,R,S')\sim\mathcal{D}}\Big[\big(R + \gamma\, V_{\bar{w}}^\Phi(S') - Q_w^\Phi(S,\phi)\big)^2\Big],$$

  with the soft state-value function

$$V_w^\Phi(s) \coloneqq \mathbb{E}_{\phi\sim\varphi_\theta(\cdot|s)}\Big[Q_w^\Phi(s,\phi) - \alpha\log\varphi_\theta(\phi \mid s)\Big].$$

We are currently investigating how to approximate the entropy in the action space.

SAC-FOP: Objectives

SAC-FOP (Action Critic)

- **Actor Objective:** The actor maximizes

$$J^{\mathrm{FOP}}(\theta) \coloneqq \mathbb{E}_{\substack{S\sim\mathcal{D} \\ \phi\sim\varphi_\theta(\cdot\mid S)}}\Big[\alpha \log\left(\varphi_\theta(\phi \mid S)\right) - Q_w^{\mathcal{A}}(S, u_0^\star(S,\phi))\Big].$$

- **Critic Loss:** The critic minimizes the soft Bellman error

$$\mathcal{L}^{\mathrm{FOP}}(w) \coloneqq \frac{1}{2}\mathbb{E}_{(S,A,R,S')\sim\mathcal{D}}\Big[\Big(R + \gamma\, V_{\bar{w}}^{\mathcal{A}}(S') - Q_w^{\mathcal{A}}(S,A)\Big)^2\Big],$$

with the soft state-value function

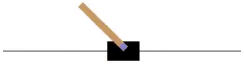
$$V_w^{\mathcal{A}}(s) \coloneqq \mathbb{E}_{\phi\sim\varphi_\theta(\cdot\mid s)}\Big[Q_w^{\mathcal{A}}(s,\, u_0^\star(s,\phi)) - \alpha \log \varphi_\theta(\phi \mid s)\Big].$$

Training: During training, we solve the MPC to obtain $a = u_0^\star(s,\phi)$ and differentiate through the control solution map to compute $\nabla_\theta J^{\mathrm{FOP}}(\theta)$.

Note: We also measure the entropy in parameter space.

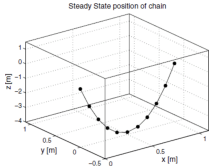
Outline of the lecture

Current Environments



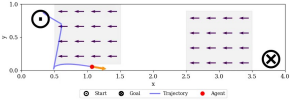
CartPole Swing-Up
The agent must swing up and balance a pole on a cart.

Goal: swing up and balance the pole



Chain
Involves stabilizing a chain of interconnected springs.

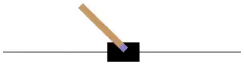
Goal: Achieve target position while minimizing oscillations

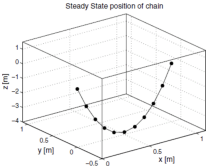


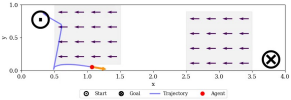
WindMaze
Point mass navigates through a maze with a spatially varying wind field.

Goal: Walk to the target while avoiding windfields, navigate through narrow passages, MPC can not see windfield

Current Environments

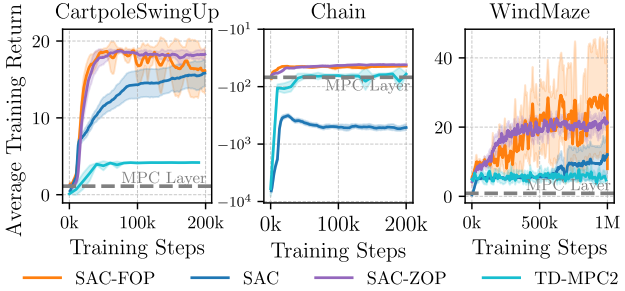






Environment	Name	Dimension	Appears in	Bounds
CartpoleSwingup	Angle reference	1	Cost	$(-2\pi, 2\pi)$
WindMaze	X-Position reference	1	Cost	$(0, 4)$
	Y-Position reference	1	Cost	$(0, 1)$
	Velocity reference	2	Cost	$(-20, 20)$
	Action reference	2	Cost	$(-10, 10)$
	$\sqrt{\cdot}$ of state residual weights	4	Cost	$(0.5, 1.5)$
Chain	$\sqrt{\cdot}$ of state residual weights	21	Cost	$(0.5, 1.5)$
	$\sqrt{\cdot}$ of action residual weights	3	Cost	$(0.05, 0.15)$

Experimental Results: Training Performance



Training Curves Across All Environments

Findings:

- **Superior sample efficiency:** Both SAC-ZOP and SAC-FOP significantly outperform vanilla SAC
- **Beats model-based RL:** Outperforms state-of-the-art TD-MPC2 baseline
- **Parameter vs Action critic:** SAC-ZOP competitive with SAC-FOP performance

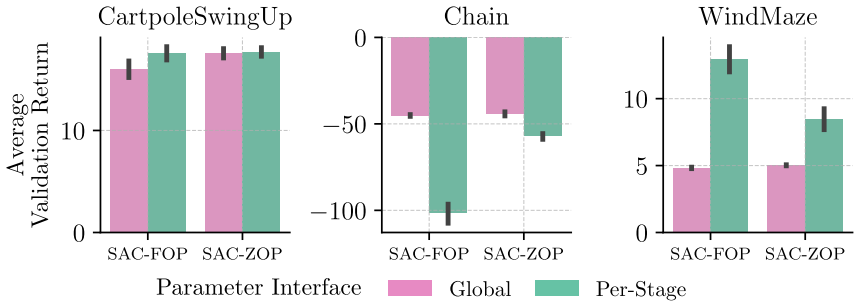
Training Time Comparison:
(1 Million steps on WindMaze)

- SAC-ZOP: **2 hours**
- SAC-FOP: 6 hours

Unsafe Exploration using Action Noise
SAC-FOA (First-Order, Action Noise), WindMaze

Algorithm	Training Violations (%)
SAC-ZOP	0%
SAC-FOP	0%
SAC-FOA	65.43%

Experimental Results: Controller Comparison



Parameter Interfaces:

- **Per-stage:** Independent parameters ϕ_k for each stage k
- **Global:** Shared constant parameter ϕ over all stages.
- Impacts both optimization complexity and the expressiveness of the control policy.

Insights:

- In some environments, flexibility helps *WindMaze*, while in other ones not *Chain*.
- SAC-ZOP remains stable even in high-dimensional per-stage setups, showing robustness of the parameter critic, e.g., *Chain* in the per-stage case has 420 parameters!

Outline of the lecture

Design Decisions
- OCP formulation
- Parameter Critic vs Action Critic
- Exploration Strategy

Algorithms: SAC-ZOP and SAC-FOP

Current Results

Next Steps

Summary

In this lecture, we...

- introduced a hierarchical RL architecture integrating an MPC prior into SAC.
- discussed key design decisions: OCP formulation, critic type, and exploration strategy.
- discussed the concept of parameter MDPs.
- presented two algorithms: SAC-ZOP (parameter critic) and SAC-FOP (action critic).
- showcased some initial results

Next Steps

1. **Theoretical Understanding**
   - Why do parameter critics work so well?
   - What are the fundamental learning dynamics?
   - When is it easier to generalize in parameter space, when in action space?

2. **More Algorithms**
   - Incorporate MPC into critic networks
   - Leverage optimal control structure for value functions
   - Use a parameter prior to improve the convergence
   - Imitation Learning

3. **Broader Applications**
   - Real-world deployment challenges
   - Heat-pump control
   - Autonomous driving

Thank you for your attention!

Background: Optimizing Parameterized Distributions

The Goal: Gradient-Based Optimization

We want to find parameters θ that optimize an objective defined as an expectation over a parameterized distribution q_θ:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{Z \sim q_\theta}[l(Z)] = \int l(z) q_\theta(z) dz$$

The gradient of this objective is:

$$\nabla_\theta \mathcal{L}(\theta) = \int l(z) \nabla_\theta q_\theta(z) dz.$$

Directly computing this integral is often intractable.

Stochastic Approximatin Theory: We don't need the exact gradient. We can still converge to a minimum if we can find a **noisy but unbiased estimator** g_t of the true gradient.

- An estimator is unbiased if $\mathbb{E}[g_t] = \nabla_\theta \mathcal{L}(\theta_t)$.
- We can then use stochastic gradient descent: $\theta_{t+1} \leftarrow \theta_t - \eta_t g_t$.

The main challenge is finding such an estimator g_t.

Background: Two Ways to Estimate the Gradient

$$\mathcal{L}(\theta) = \mathbb{E}_{Z \sim q_\theta}[l(Z)]$$

REINFORCE (Log-Derivative Trick)

Rewrite the gradient as an expectation that can be sampled:

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{Z \sim q_\theta}\left[\underbrace{l(Z)}_{\text{The Loss}} \underbrace{\nabla_\theta \log q_\theta(Z)}_{\text{The "Score Function"}} \right]$$

The Monte Carlo estimator is:

$$g_t' \approx \frac{1}{N} \sum_{n=1}^{N} l(Z_n) \nabla_\theta \log q_{\theta_t}(Z_n)$$

- ✓ Broadly applicable.
- ✗ Often suffers from high variance in empirical RL

Reparameterization Trick

Factor out the randomness. Express the sample Z as a differentiable function of a noise variable ξ:

$Z = f(\theta, \xi)$, where $\xi \sim q_0$ (e.g., standard normal)

Now the gradient can be moved inside the expectation under some regularity conditions:

$$\nabla_\theta \mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\xi \sim q_0}\left[\nabla_\theta l(f(\theta, \xi)) \right]$$

The Monte Carlo estimator is:

$$g_t'' \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta l(f(\theta_t, \xi_n))$$

- ✓ In empirical RL lower variance, often more stable.
- ✗ Only applicable when f is known, continuous, and differentiable almost everywhere.

Background: Parameter MDP

Idea: Learn policy φ over parameters $\phi \in \Phi$ instead of a policy π over actions $a \in \mathcal{A}$.

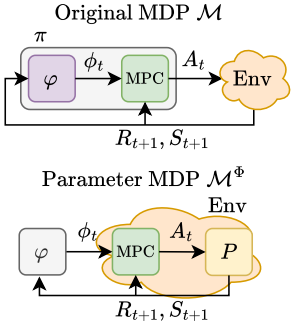
Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r)$ be the original MDP with states $\mathcal{S}$, actions $\mathcal{A}$, transition kernel P, and rewards r.

Definition: Parameter MDP ($\mathcal{M}^\Phi$)

New MDP $\mathcal{M}^\Phi = (\mathcal{S}, \Phi, P^\Phi, r^\Phi)$ with:

- **Actions**: Parameter space Φ
- **Transitions**: $P^\Phi(\cdot \mid s, \phi) \coloneqq P(\cdot \mid s, u_0^\star(s, \phi))$
- **Rewards**: $r^\Phi(s, \phi) \coloneqq r(s, u_0^\star(s, \phi))$

Assumption: $u_0^\star$ is a deterministic, total function



New objects: Parameter policy $\varphi(\phi \mid s)$, induced action policy $\pi(a \mid s) = \mathbb{E}_{\phi \sim \varphi}[\mathbb{1}_{a = u_0^\star(s,\phi)}]$, and corresponding value functions $Q^\varphi(s, \phi)$ and $Q^\pi(s, a)$. We have that

$$Q^\varphi(s, \phi) = Q^\pi(s, u_0^\star(s, \phi))$$

SAC Pseudocode

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \Big(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\Big),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence