Goal-Conditioned Generators of Deep Policies

Francesco Faccio*, Vincent Herrmann*, Aditya Ramesh, Louis Kirsch, Jürgen Schmidhuber
{francesco.faccio, vincent.herrmann, aditya.ramesh, louis.kirsch, juergen}@idsia.ch

Istituto Dalle Molle di studi sull'intelligenza artificiale (IDSIA, USI/SUPSI), KAUST

@FaccioAI
@idivinci
@LouisKirschAI
@SchmidhuberAI

- Goal-conditioned Reinforcement Learning (RL) learns optimal policies that satisfy commands.
- We consider commands of the form "act in the environment to achieve a desired expected return".
- Methods based on hindsight learning (like Upside-Down RL) maximize likelihood of the agent's behavior given reward commands.
- **Problem**: Many behaviors satisfy the same command, maximum likelihood estimation may fail.
- **Solution**: Learn an evaluator function mapping policy parameters to expected return. For any return command, generate policy weights that match the evaluator's prediction.

arxiv paper: arxiv.org/abs/2207.01570

Method

Fast Weight Programmers (FWPs)

- FWPs are neural networks that generate weights of another neural network
- They can be conditioned on some contextual input

In a Markov Decision Process with:

- Policy π_θ,
- $p(\tau|\theta)$: distribution over trajectories τ induced by π_θ
- Context information $c \in \mathbb{R}^{n_c}$
- Deterministic FWP function $G_\rho : \mathbb{R}^{n_c} \longrightarrow \Theta$
- Probabilistic generator $g_\rho(\theta|c) = G_\rho(c) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, fixed σ

Given a return command c, we learn a generator that produces policies whose return is c. Our objective is to enforce the following equality for all c:

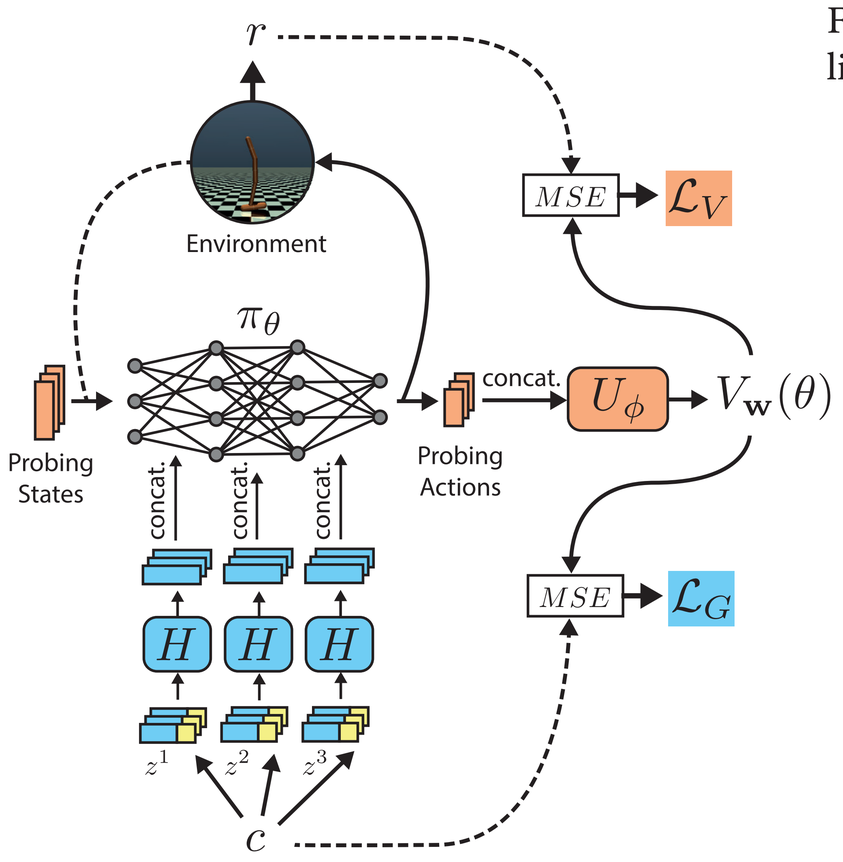
$$J(\rho, c) = \int_\Theta g_\rho(\theta|c) \int_\mathcal{T} p(\tau|\theta) R(\tau) \mathrm{d}\tau \mathrm{d}\theta = c$$

Parameter-based Evaluator

- To make the objective differentiable, we learn an evaluator function $V_\mathbf{w} : \Theta \longrightarrow \mathbb{R}$
- $V_\mathbf{w}(\theta)$ estimates the expected return of policy π_θ using supervised learning

- The generator should be instructed to produce larger and larger returns to ensure that the generated policies improve over time

Algorithm



For a Generator that produces policies achieving any given return:

repeat until convergence:

- Choose command c using some strategy
- Generate policy parameters $\theta = G_\rho(c) + \epsilon$ (Gaussian ϵ)
- Generate episode in environment with policy π_θ
- Save return r and parameters θ in buffer D
- Update Evaluator by SGD: $\nabla_\mathbf{w} \mathbb{E}_{(r,\theta)\in D}[(r - V_\mathbf{w}(\theta))^2]$
- Update Generator by SGD: $\nabla_\rho \mathbb{E}_{r\in D}[(r - V_\mathbf{w}(G_\rho(r)))^2]$

Policy Fingerprinting

Architecture of the Evaluator $V_\mathbf{w}$

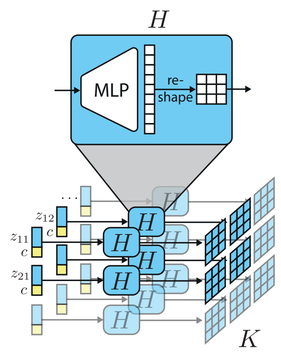
- A set of learnable probing states is given as input to the policy π_θ
- The resulting outputs of the policy are called probing actions
- They are given as input to a multi-layer perceptron U that computes $V_\mathbf{w}(\theta)$

- Probing states learn to query the policy in meaningful situations, so that the policy's success can be judged by its probing actions
- Probing actions can be seen as a policy embedding

HyperNetworks

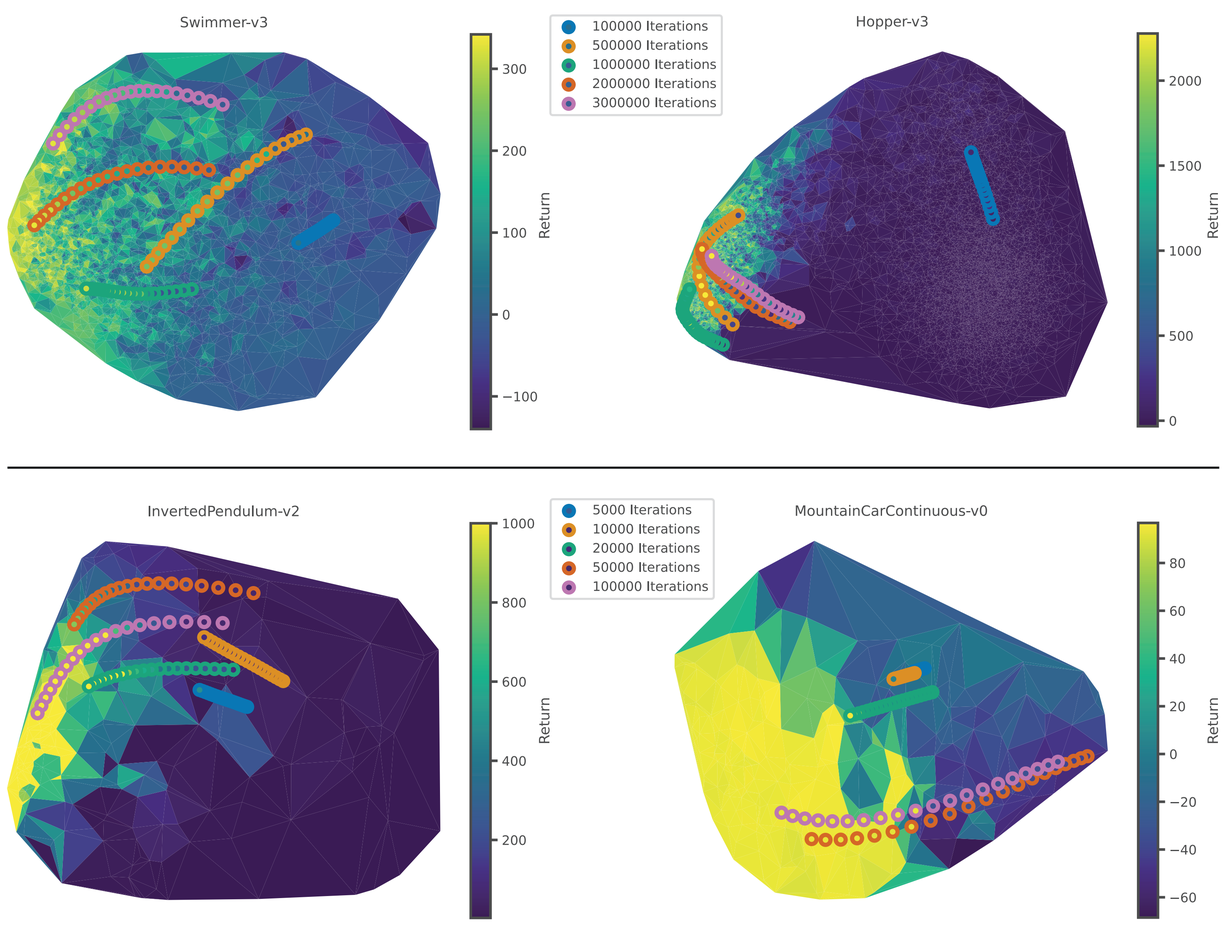
Architecture of the Generator G_ρ

- Shared neural network H_ξ receives as inputs z (learnable embedding) and return command c
- Outputs slice s of a weight matrix: $s^j_{mn} = H_\xi(z^j_{mn}, c)$ for layer j and coordinates of the slice m and n
- c and the learned slice embedding z^j_{mn} are concatenated and given as input to H



Experiments

- Evaluation on continuous control tasks from the MuJoCo suite
- All policies are multi-layer perceptrons with two hidden layers, each having 256 neurons

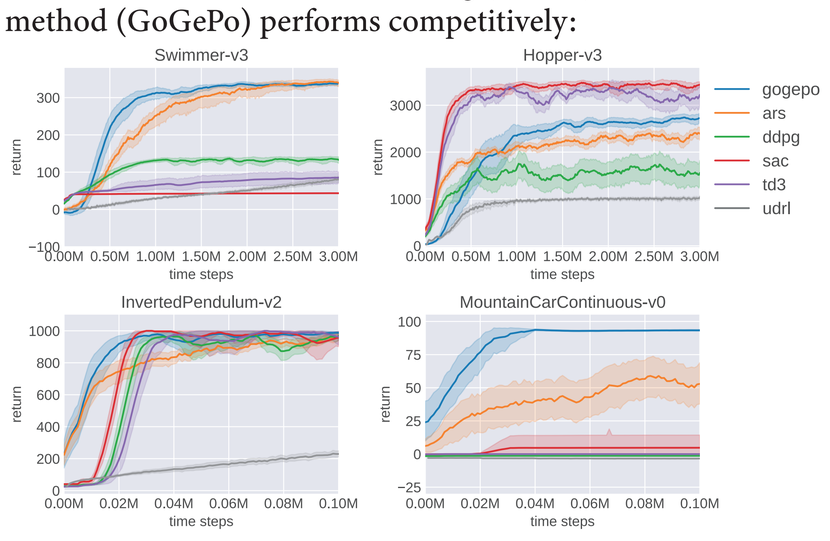


PCA applied to probing actions to show all policies in the buffer after training, as well as policies created by the generator at different stages of training when given the same range of return commands. Policies are colored in line with achieved return. The generator is able to produce policies across the whole performance spectrum.

Baselines:

- Augmented Random Search (ARS)
- Deep Deterministic Policy Gradients (DDPG)
- Soft Actor-Critic (SAC)
- Twin Delayed DDPG (TD3)
- Upside Down Reinforcement Learning (UDRL)

Our method (GoGePo) performs competitively:



Strategy for choosing the next command:

- The simple strategy "produce a policy whose return is 20 above the one of the best policy seen so far" can be very effective (see ablation below)
- Our method learns to understand and exploit the nature of performance improvements in a given environment