I(MoE)-SAC: Interpretable Mixture of Interpretable Experts via Soft Actor-Critic for Inverse Scaling in Continuous Control

M. Vincze, L. Ferrarotti, L. L. Custode, B. Lepri, G. Iacca

1 I(MOE)-SAC MODEL

In order to tackle the challenge posed by the Walker2D environment from MuJoCo [3] while retaining interpretability, I(MoE)-SAC relies on the training of a policy structured according to a Mixture-of-Experts (MoE) architecture

$$\pi(s) = \sum_{m=1}^{M} [\, g(s \mid \theta)\,]_m \; \pi_m(s \mid \theta_m) \tag{1}$$

that is a combination of:

- A set $\{\, \pi_m(\cdot \mid \theta_m) : \mathcal{S} \to \mathcal{A} \mid m = 1, \ldots, M \,\}$ of M parameterized continuous policy functions with state space $\mathcal{S} = \mathbb{R}^{n_s}$ as domain, and action space $\mathcal{A} = \mathbb{R}^{n_a}$ as co-domain, representing M decision makers, trained to be experts in different skills.
- A parameterized router $g(s \mid \theta) : \mathcal{S} \to [\,0, 1\,]^M$, structured as $g(s \mid \theta) = \mathrm{TOP}_1(\,\mathrm{softmax}(\,\hat{g}(s \mid \theta)\,)\,)$. The softmax function transforms $\hat{g}(s \mid \theta) \in \mathbb{R}^M$ in a preference vector, such that its m-th element measures the preference in choosing the m-th expert to act in state s. The function TOP_1 assigns value 1 to the vector component with maximum preference, and 0 to the others. We indicate as $[\, g(s \mid \theta)\,]_m$ the m-th component of vector $g(s \mid \theta)$.

This architecture takes inspiration from [2], where MoE neural layers are employed in computer vision for expert detection of sub-patterns in images through a sparse network. Here, such architecture is tuned to retain interpretability in continuous control, by removing the non-linearities and activation functions, employing a single layer, and shaping the router and the experts π_m and $\hat{g}$ as linear functions, that is: $\pi_m(s \mid \mu_m) = \mu_m \cdot s, \quad \hat{g}(s \mid \theta) = \theta \cdot s$, with $\mu_m \in \mathbb{R}^{n_a \times n_s}$ and $\theta \in \mathbb{R}^{M \times n_s}$.

1.1 Policy training

The parameterized functions are trained via Reinforcement Learning. We ensure to explore the action space of each expert π_m by injecting stochasticity in the choices in training, i.e., $\pi_m(s \mid \mu_m, \sigma_m) = \mathcal{N}(\,\mu_m \cdot s,\ \sigma_m^2\,)$. We rely on Soft Actor-Critic (SAC) [1], a state-of-the-art algorithm for continuous control, that balances the objective function with a term promoting higher entropy policies, for exploration. In order to ensure workloads are balanced among the M experts, we augment the SAC objective function with additional losses, introduced in [2]. Moreover, we add exploration noise to the router, i.e. $\hat{g}(s \mid \theta) = \theta \cdot s + \epsilon$, with $\epsilon \sim \mathcal{N}(0,\ 1/M^2)$. In order to achieve statistical reliability in our results, we perform $N_{\text{train}} = 10$ independent training runs. The episodic rewards (ER) achieved in training are represented in Figure 1, where we plot the mean and standard deviation of episodic returns evolution over one million environment interactions. In the same plot, we highlight the performance of the policy that we later interpret, indicated as $\pi^\star$. We assign the ER to all the steps of the same episode, and we plot the ER at each timestep (that, is at each interaction with the environment), to make episodes with different lengths comparable. On average

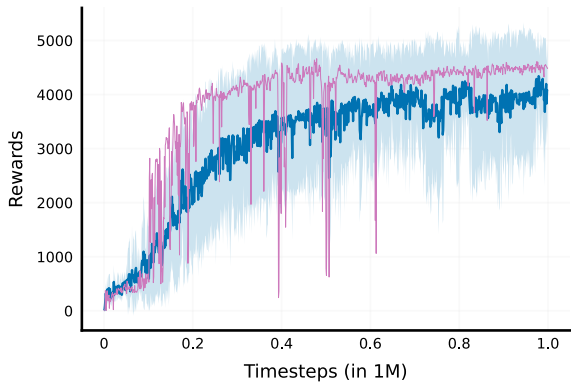

Figure 1: ER in training: mean and standard deviation over $N_{\text{train}} = 10$ runs (in blue), and ER of $\pi^\star$ (purple line).

Table 1: Performance in testing ($N_{\text{test}} = 100$ scenarios)

	ER mean	ER std	ER max	ER min	% success
$\pi^\star$	4593.3	61.1	4716.8	4469.7	100 %
avg	3922.5	322.2	4209.8	2812.3	89.4 %

our model, while being interpretable, reaches higher scores quicker than a neural policy trained with SAC, and matches it in final performance [1]. This is true in particular for $\pi^\star$, which presents some drops in performance, during the initial part of the training, but then stabilizes on high score results. This behavior, typical of the learning phase, gets averaged out in the statistics in Figure 1.

2 TESTING AND PERFORMANCE

We test the learned parameters $(\,\theta,\ \{\,\mu_m \mid m = 1, \ldots, M\,\})$, while not employing $\{\,\sigma_m \mid m = 1, \ldots, M\,\}$, and hence obtaining a deterministic policy, as it is often done with SAC [1]. Here we include the results achieved on average over $N_{\text{test}} = 100$ random scenarios.

2.1 Scoring performance

Table 1 includes ER mean, standard deviation, minimum, and maximum, plus success rate, achieved in testing. We evaluate both the average scores among the 10 trained policies, and we highlight the ones of $\pi^\star$. Again, our model matches the performance of nonlinear SAC observed at the end of training [1], while employing a policy with 100-times less parameters, and no activation functions. Hence we obtain an interpretable and well-performing policy while simplifying the policy structure and reducing computational cost.

2.2 Interpretation

In this section the trained router and the experts characterizing $\pi^\star$ are interpreted. We will sustain our analysis with two figures: Figure 3 shows the average input fed to each expert and its corresponding average action, while Figure 2 includes the relevant

Figure 2: Learned weight matrices μ_m for the three active experts and corresponding router weights $\theta(m, \cdot)$

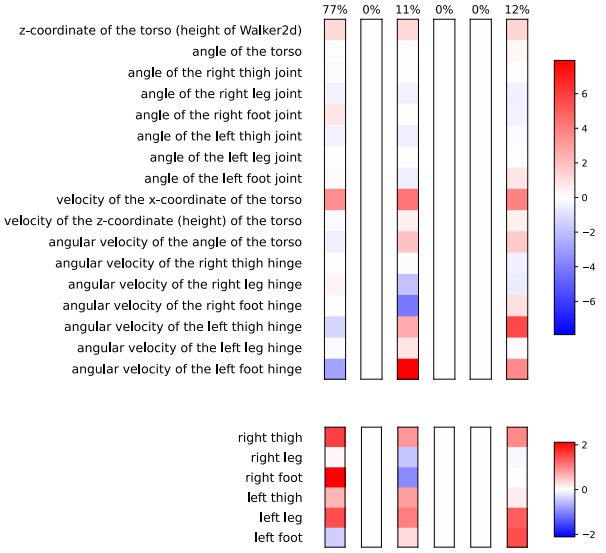

Figure 3: Top diagram: components of the average input state assigned to each expert. Bottom diagram: components of the average output (action) produced by each expert.

trained weight matrices of the experts and the corresponding row-vectors of the router. With respect to this figure, we will only interpret the weights with the highest magnitudes. We observe that all experts mainly use the first ten features of the observation space, therefore ignoring the angular velocities.

Router. As visible from Figure 3, the router employs only experts 1, 3, and 6, after learning through exploration that it is possible to solve the task using only 3 experts over 6. The three experts distribution (i.e., the rate of times each expert is used) is not uniform. This is reasonable given that the learned policy requires a lot of repeated actions, achieved using a single expert. Observing the router vectors $\theta(\cdot, m)$ included in Figure 2, it is clear that the router uses only two features to separate the observation space into regions where a single expert is sufficient: "angle of the right foot" and "angle of the left thigh". In particular, the routing score of Expert 1 is positively proportional to both the features, the one of Expert 3 shows a negative relation to them, and the one of Expert 6 shows a negative relation exclusively with the angle of the right foot. The descriptions of the average states routed to different experts are included in the following experts explanations.

Expert 1. The first expert receives observations where the angular velocities of the left thigh and foot are lower than the average (jump using left leg is finished). Its corresponding action is to strongly accelerate with all rotors except the right leg (not used) and left foot (see Figure 3). This action reinforces the running motion by accelerating most of the rotors. In particular, it slows down the left foot using strong positive correlations with most features except "angle of the torso", while it uses strong negative correlations with features such as "angle of the torso" and "angle of the right leg" and strong positive correlations with features such as "angle of the left thigh" and "height of Walker2d" to output the rest of the actions (see Figure 2).

Expert 3. The second expert is queried with observations where the angular velocity of the right foot is significantly below, and the angular velocity of the left foot is significantly higher w.r.t. the average. The mean action executed by this expert induces a strong negative torque in the right leg and the right foot, while accelerating the whole left leg. This indicates that the agent is trying to make "bumps" with the right foot and leg, while walking more smoothly with the left leg (see Figure 3). To calculate the actions, it uses strong negative correlations with features such as "angle of the torso" and "angle of the right leg" and strong positive correlations with features such as "angle of the left thigh" and "height of the torso". The decision to move the right foot has a strong, negative correlation with the angle of the torso, indicating that it uses the right foot to balance the torso's angle quickly. Moreover, we also observe a smaller dependency w.r.t. the left thigh. This confirms that the agent has a bumpy gait, as it aims to balance the torso with much stronger torques than the previous expert (see Figure 2).

Expert 6. This expert receives observations with a high angular velocity of the left thigh and in general no negative features. It has similar weights to Expert 3 with stronger correlations to features such as "angle of the right foot", "angle of the left thigh" and "angle of the left foot". As it receives mostly positive features, its weights have many strong negative correlations on the columns corresponding to the right leg related actions. These observations from both figures show that Expert 6 outputs actions related to stepping with the left leg.

REFERENCES

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[2] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *CoRR*, abs/2106.05974, 2021.

[3] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.