Learning Environmental Calibration Actions for Policy Self-Evolution

Chao Zhang, Yang Yu, Zhi-Hua Zhou  
emails: {zhangc, yuy, zhouzh}@lamda.nju.edu.cn
National Key Laboratory for Novel Software Technology, Nanjing University

1. Motivation

Physical world reinforcement learning is highly costly

Simulators are usually very helpful

Simulation error is inevitable:
- measure inaccuracy
- physical world changes

Previously: simulated policy + manual adjustment

Can the policy be self-evolvable to adapt to its environment?

2. Idea

Learn a meta-policy over environments

1. collect a set of previous environments
2. associate policies with environments
3. learn a mapping from environment features to policy parameters

such policy is generalizable to new environments

In a new environment: map the environmental features to the policy

But, how to obtain the environmental features?

3. Proposed Method: POSEC

Framework

Extraction of environmental features:
1. run some calibration actions
2. observe the environment states after each action
3. the observations are used as the features

With the environment features, aggregate previous policies for the new environment

Implementation

Assume a configurable simulator is available to generate environments

Sample \( M \) different configurations of the environment
In every environment: train a policy heavily, as a base policy

\[ \text{base policies set: } \{ \pi_1, \pi_2, \ldots, \pi_M \} \]

\[ \text{combination weights} \]

linear combination policy:

\[ \pi_w(a|s) = \sum_{i=1}^{M} \frac{w_i}{\sum_{j=1}^{M} w_j} \pi_i(a|s) \]

Draw another set of \( M \) different configurations of the environment
In each environment: the reward objective function about \( \pi_w(a|s) \)

\[ J_{MDP}(w) = \int P_R(r) R(r) \, dr \]

Optimal weights:

\[ w^*_i = \arg \max_w J_{MDP}(w) \]

solved by a derivative-free optimization method (Yu et al., IJCAI’16; Hu et al., AAAI’17)

solved combination weights (\( w_1, w_2, w_3, \ldots, w_M \) of base policies)

regression model

In each of the \( M \) environment:

run the calibration actions \( A \) (assume given)

the environmental features

combination weights

instance  \( \text{regression model} \)  \( \text{label} \)

4. Experiments

Experimental task (State 23 dim, Action 7 dim)

Robotic arm controlling to accomplish tasks:
- Pusher: pushes a cylinder onto a coaster
- Striker: hits a ball to a target
- Thrower: throws a ball into a box

Taking the Pusher task as an example

Three policy learning methods in the new environment

Experimented algorithms
1. The policy trained directly from scratch
2. The policy trained using LSTM for environment adaptation
3. The policy evolved by POSEC

Comparisons of performance in new environments (TRPO+LSTM need to be trained online)

Comparison of refinement training from different initial policies.

source codes can be found at:  
https://github.com/eyounx/POSEC

demo video: