Appendix
Rediscovering Affordance: A Reinforcement Learning Perspective

YI-CHI LIAO, Aalto University, Finland
KASHYAP TODI, Aalto University, Finland
ADITYA ACHARYA, Aalto University, Finland and University of Birmingham, United Kingdom
ANTTI KEURULAINEN, Aalto University, Finland
ANDREW HOWES, Aalto University, Finland and University of Birmingham, United Kingdom
ANTTI OULASVIRTA, Aalto University, Finland

A MODEL

The model consist of two independent components – a *policy-model* and a *motion classifier*. The *policy-model* represents the control knowledge, which is a mapping between the states and actions. To train the agent we use a type of *policy gradient method* called Proximal Policy Optimization (PPO) [2]. PPO is a type of policy search algorithm that solves the problem of finding the optimal policy by representing the policy space in a parametric form and then using a sampling-based technique to find the optimal policy parameters.

In policy search algorithms, the goal is to find optimal parameters such that it maximizes some future cumulative expected reward over a finite horizon T. Let, J(θ) be the objective function of policy π parameterised by θ.

$$\max_{\theta} J(\theta)$$

$$J(\theta) = \mathbb{E}_{\theta}^{\pi}\Big[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\Big] \tag{1}$$

Where, To solve equation 1, *policy gradient* is the most popular approach used. The idea here is to shift the parameter vector θ (e.g., θ can be the weights of a neural network) by a small amount by calculating the gradient of $\nabla J(\theta)$ such that an optimal θ is found. The gradient is defined using the likelihood-ratio theorem. Here the expected reward in each state is the weighted sum of the probability of being in the state and the reward.

$$\mathbb{E}_{\theta}^{\pi}\Big[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\Big] = \sum_{t=1}^{T} P_{\theta}(s_t, a_t)\mathcal{R}(s_t, a_t) \tag{2}$$

$$\nabla J(\theta) = \mathbb{E}_{\theta}^{\pi}\Big[\Big(\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\Big)\nabla\Big(\sum_{t=1}^{T} logP_{\theta}(s_t, a_t)\Big)\Big] \tag{3}$$

While equation 3 is an unbiased estimator of gradients for the expected cumulative reward, it still has high variance. In other words, the optimizer may take many steps in the poor direction, even though, on average, it will end up in the correct direction. This may lead to weak or slow convergence. Alternatively, any constant baseline value that is independent of the action can be subtracted from the gradient to minimize variance.

$$\nabla J(\theta) = E_{\theta}^{\pi}\Big[\nabla \sum_{t=1}^{T} logP_{\theta}(s, a)\Big(\sum_{k=t}^{T} \mathcal{R}(s_k, a_k) - b(s_t)\Big)\Big] \tag{4}$$

Table 1. Hyperparameters used for the policy model.

parameters	value
learning rate	0.00008
horizon (maximum steps per episode)	1000
nsteps (number of steps per policy update)	6000
minibatch	2000
hidden layers in the policy network	5
hidden layers in the value network	5
nodes per layers	128
activation function	tanh
optimiser	Adam

Table 2. Hyperparameters used for the motion classifier.

parameters	value
learning rate	0.005
epochs	250
hidden layers	2 (1 LSTM, 1 MLP)
hidden dimension	32
nodes per layers	32
activation function	tanh
optimiser	Adam

PPO falls in the family of the Actor-Critic algorithm. Here, the algorithm uses the state value function $V(s)$ as a baseline. In the Actor-Critic algorithm, the Critic is responsible for calculating the value function $V(s)$ following policy π. The Actor then uses the value function estimated by the Critic and uses equation 4 to update the policy parameters. PPO additionally improves training stability by avoiding parameter updates that change the policy too much at one step. It carries out this idea by restricting the search inside a trust region of the policy used in the data collection. In our implementation, the policy and value networks are separate networks without parameter sharing, and both of them have 5 layers of multilayer perceptron (MLP).

The *motion classifier* is described as a supervised learning model. It uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized. For the *motion classifier*, the training set consists of a sequence of actions generated by the policy model. It classifies the sequence of actions generated by the *policy-model* and ground truth about the motion labels provided by the modeler. *Motion classifier* model uses a neural network with a recurrent network component; specifically, an LSTM [1] to accumulate information throughout the episodes. We use a recurrent network to account for the sequence each action is performed in with cross-entropy loss. The structure in our implementation is a single LSTM layer followed by an MLP layer to form a 2-layer neural network.

REFERENCES

[1] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (11 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 arXiv:https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347