**Supplementary Material:**
Progressive Relation Learning for Group Activity Recognition

Guyue Hu\(^{1,3*}\)  Bo Cui\(^{1,3}\)  Yuan He\(^{1,3}\)  Shan Yu\(^{1,2,3}\)

In this supplementary material, (1) we firstly introduce some details about the standard flowchart of the A3C algorithm [3]. (2) We then take the proposed FD agent as an example to show how to specify the state, action and reward in the standard flowchart A3C algorithm. (3) In addition, we provide additional result and analysis about the confusion matrix on CAD dataset [1].

1. **The details of A3C algorithm [3]**

The asynchronous advantage actor-critic (A3C) algorithm [3] is a classical optimization algorithm for reinforcement learning, which contains many asynchronous updating Actor-Critic agents [2]. In A3C algorithm, there are numerous agent copies (workers) and each of them runs in an independent CPU thread/core. Each agent copy (worker) accumulates gradients to update a set of the globally shared parameters (including the value function parameters \(\theta_v\) and the policy parameters \(\theta\)). The global updating occurs when any worker runs every \(\tau_{\text{max}}\) (updating interval) steps or reaches a terminal state. Therefore, the running process of every worker (CPU thread/core) is relatively independent, and the globally shared parameters are updated asynchronously by these workers.

Specifically, the agent copy (Actor-Critic agent) in each thread maintains both a thread-specific policy \(\pi(A_{\tau}; S_{\tau}; \theta')\) (also named Actor) to generate actions and a thread-specific estimation of value function \(V(S_{\tau}; \theta'_v)\) (also named Critic) to assess values for corresponding states. The accumulated reward at the step \(\tau\) is \(R_{\tau} = \sum_{i=0}^{k-1} \gamma^i r_{\tau+i} + \gamma^k V(S_{\tau+k}; \theta'_v)\), where \(\gamma\) is the discount factor, \(r_{\tau}\) is the reward at the \(\tau\)th step, and \(k\) varies from 0 to \(\tau_{\text{max}}\). In addition, the objective functions of A3C algorithm are formed from the advantage function [3] and entropy regularization term [4]. The advantage function can be calculated by \(R_{\tau} - V(S_{\tau}; \theta'_v)\), and the entropy of policy \(\pi\) is denoted as \(H(\pi(S_{\tau}; \theta'))\). Once a global updating is triggered, the gradients of Critic are accumulated via Eq. S1 to update the **globally shared value function parameters** \(\theta_v\). Similarly, the gradients of Actor are accumulated via Eq. S2 to update the **globally shared policy parameters** \(\theta\).

\[
d\theta_v ← d\theta_v + \nabla_{\theta_v} (R_{\tau} - V(S_{\tau}; \theta'_v))^2 / 2, \quad \text{(S1)}
\]

\[
d\theta ← d\theta + \nabla_\theta \log \pi(A_{\tau}; S_{\tau}; \theta') (R_{\tau} - V(S_{\tau}; \theta'_v)) + \beta \nabla_\theta H(\pi(S_{\tau}; \theta')) \quad \text{(S2)}
\]

where \(\beta\) controls the strength of the entropy regularization term. After global updating at this time, the local agent synchronizes its parameter values from the updated globally shared parameters, i.e.

\[
\theta' = \theta, \theta'_v = \theta_v. \quad \text{(S3)}
\]

Finally, the local agent clears its gradient caches and starts to accumulate new gradients again. Because there are numerous local agent copies in many independent CPU threads, the trigger times of global updating are completely determined by the running speed of each thread.

The most important contribution of A3C algorithm is introducing the **Asynchronous Updating** policy. The policy involves numerous independent workers, whose implementation relies on the numerous threads of multi-core CPU. As stated by [3], although it works on a multi-core CPU, it achieves better results, in far less time than previous GPU-based algorithms, using far less resource than massively distributed approaches.

2. **Training flowchart for the FD agent**

The proposed feature-distilling (FD) agent and relation-gating (RG) agent are both optimized by the A3C algorithm [3]. The standard flowchart of A3C algorithm is introduced above, and we can obtain the detailed pipeline of agent training stage by simply specifying the state, action, and reward defined for each agent in the main text. The detailed pipeline of FD agent is shown in the Algorithm S1, only one thread-specific agent is shown to illustrate a minimal flowchart. In practice, we apply 16 threads (workers) to asynchronously update the globally shared parameters, and we stop all the threads once any thread have run 2 hours (MaxRuntime). The stop command is controlled by a globally shared variable, Runtime, which records the longest runtime of all the threads. The runtimes of threads are obtained by the public Python module time (https://docs.python.org/3/library/time.html).
### 3. Additional Confusion Matrix

To better understand the performance of the proposed framework, the additional confusion matrix of our PRL on the CAD dataset [1] is shown Fig. S1. The results clearly show that our PRL has nearly solved the recognition of activities “Queuening” and “Talking”, which proves the effectiveness of the proposed framework. However, the class “Waiting” is seriously confused by the class “Moving” probably because some “Moving” activities under the scene of crossing street are ill co-occur with some “Waiting” activities in the CAD dataset. Data cleaning or more training data are required to distinguish these two categories.
Figure S1: Confusion matrix on the CAD dataset.

References


