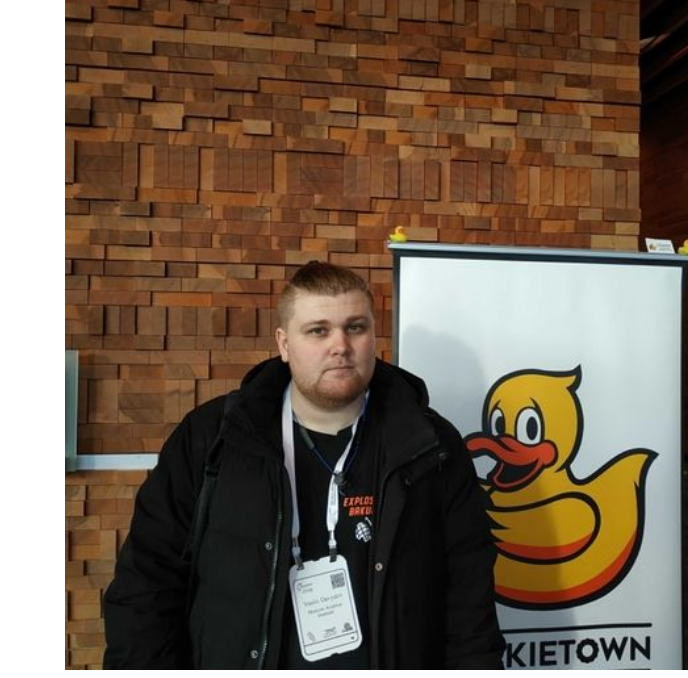


#117



Self and Other Modelling in Cooperative Resource Gathering with Multi-Agent Reinforcement Learning.

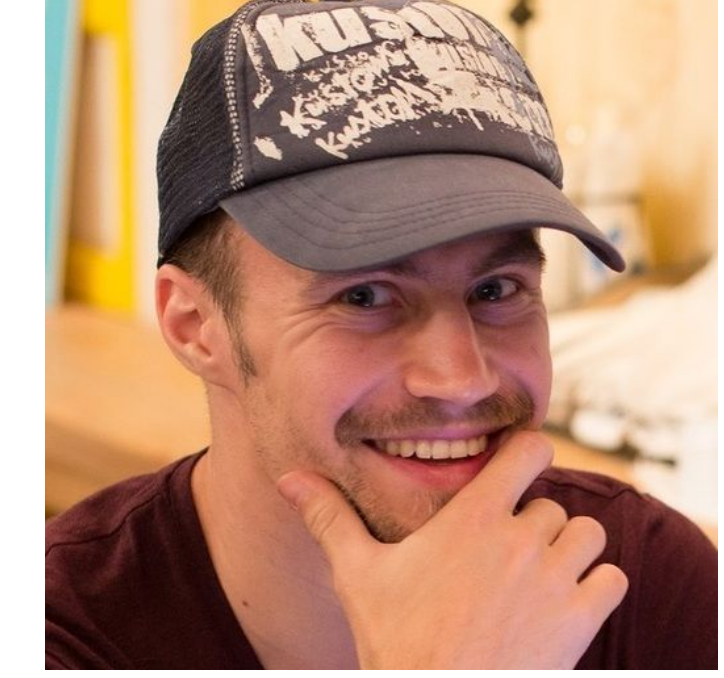
Vasilii D. Presenter^{*1}, Timofei L. ², Aleksandr P.³. ¹Moscow Aviation Institute, Moscow, Russia vddavydov@mai.education ²Moscow Institute of Physics and Technology (National Research University), Moscow, Russia tim.luska@protonmail.com ³Artificial Intelligence Research Institute FRC CSC RAS, Moscow, Russia



Vasilii Davydov



Timofei Liusko



Aleksandr I. Panov

INTRODUCTION

In cooperative RL problems it is crucial to achieve agents to behave cooperatively.

It is also important to prevent the information exchange between the agents so that they could act independently.

PROBLEM SETUP

In this work, we explored the application of the Self-Other Modelling algorithm on several multi-agent reinforcement learning algorithms. A3C agent architecture was compared to more complicated Hide and Seek architecture. The experiments were set in the grid-based environment. The goal was cooperatively collect coins; from three colors, agents were required to collect coins of particular two colors and avoid collecting coins of the third color.

SOM ALGORITHM

Self other modelling algorithm (SOM) - is an algorithm in which the agents approximate vector of goals of an other agent while acting in an environment. Agents get a set <state S, self goal vector Zself and other agent's goal vector Zother> as input and returns action distribution. Every step in the environment agents update Zother vector with the following loss function:

$$CE(\pi(S_{other}, z_{other}, z_{self}; \theta_{self}), a_{other}^i)$$

A3C ALGORITHM

Asynchronous advantage actor critic algorithm (A3C) - is an algorithm for updating agent's weights. In the end of every episode an agent gets reward R. A discounted reward is then calculated

$$R_i = r_i + \sum_{j=1}^k r_{i+j} * \gamma^j$$

With discounted reward advantage function is calculated

$$A(s_i) = R_i - V(s_i)$$

Then with the advantage function policy loss function and value loss function are calculated

$$L_{policy} = -\log(p(a_i|s_i))A(s_i)$$

$$L_{value} = A(s_i)^2$$

Also for developing the agent's exploration we add entropy loss function. It orces agents to avoid action distributions with one dominant action

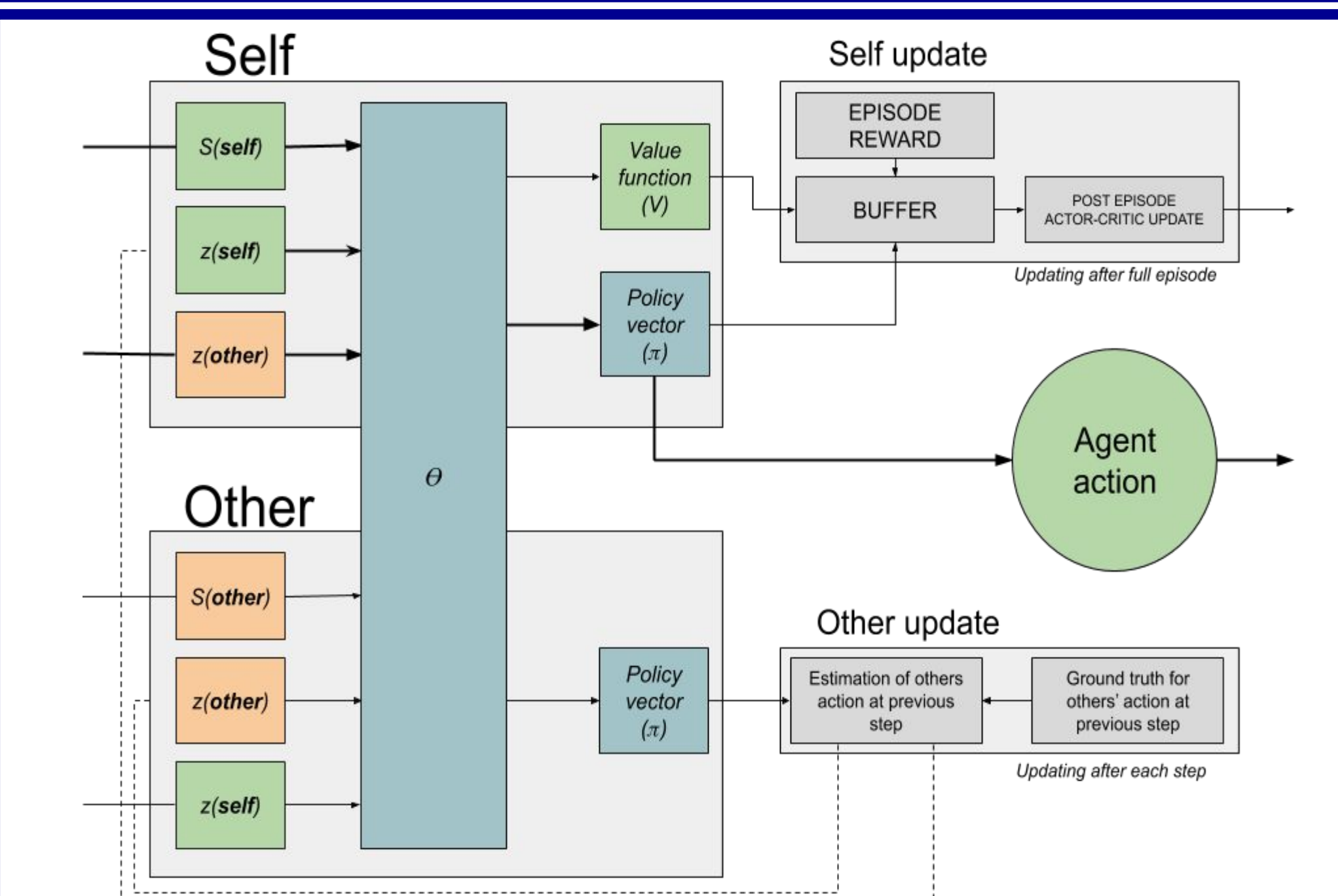
$$L_{entropy} = -\sum p(a_i|s_i) \log(p(a_i|s_i))$$

Final A3C loss function is calculated with all three loss functions

$$L_{A3C} = L_{policy} - \beta L_{entropy} + \alpha L_{value}$$

This loss is then averaged and used to update the agent network

AGENT ARCHITECTURE

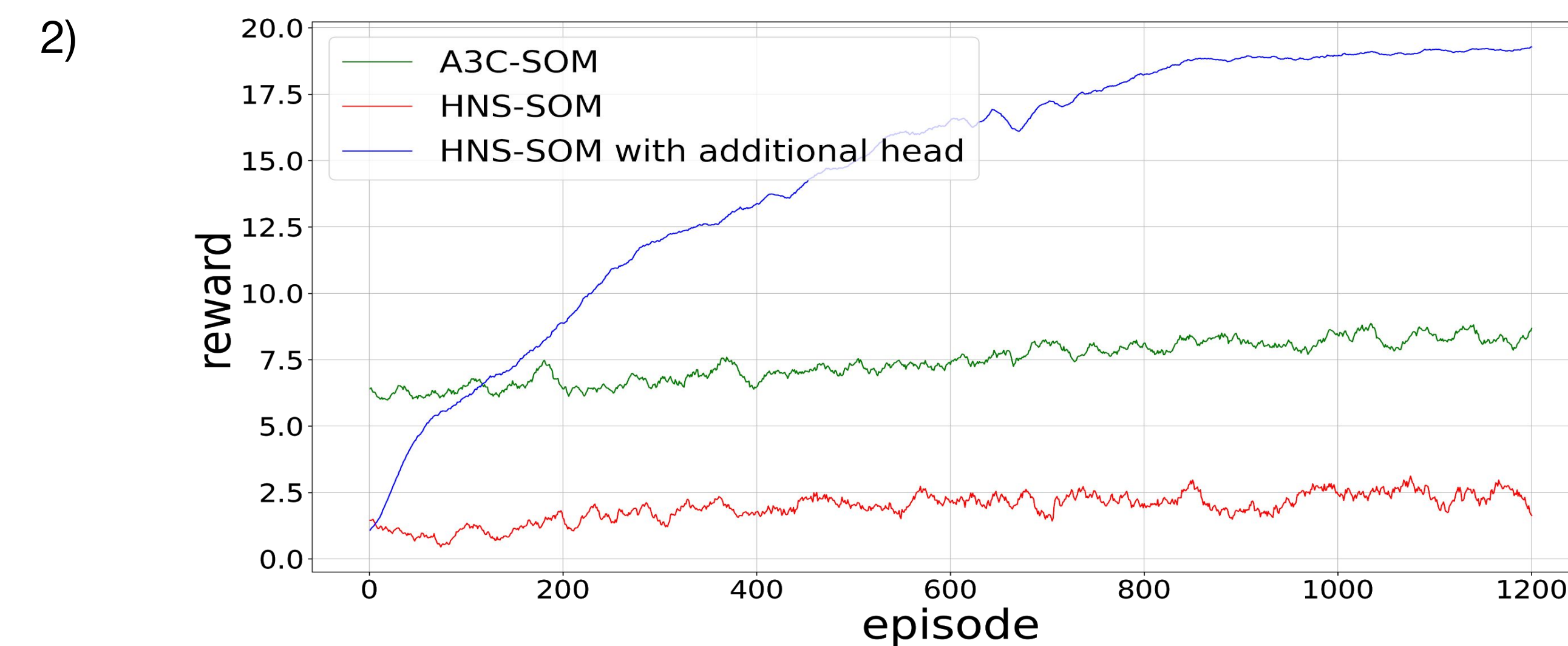
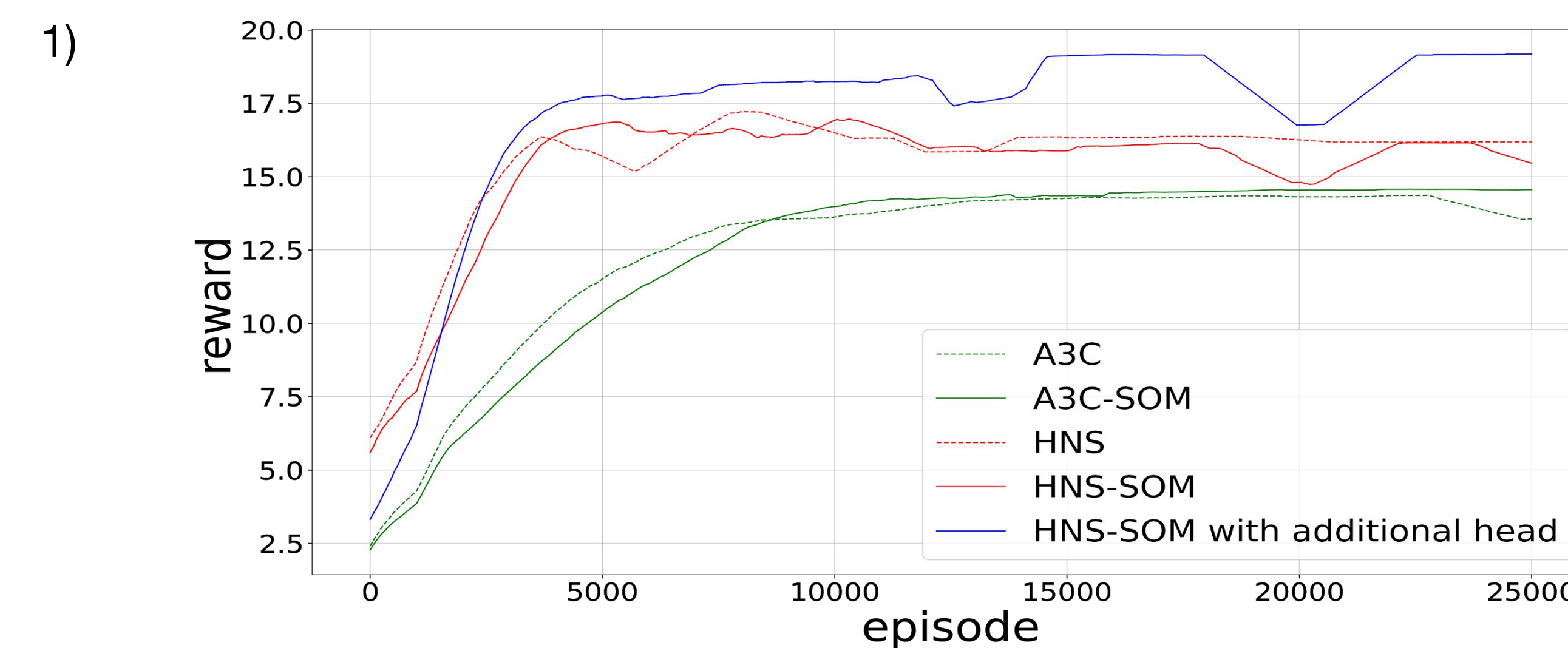


θ ARCHITECTURE



Base A3C(left), Hide and seek(right)

RESULTS



ANALYSIS

The first series of experiments was set in the environment with 5 fixed seeds.

- A3C and A3C-SOM achieved reward 14
- HNS and HNS-SOM achieved reward 16
- HNS-SOM with additional head achieved reward 18

HNS agents experienced downfalls in the reward during the learning process. It can possibly be explained that the seed-initialization offered possible shortcuts to a local minimum, with not enough variety for agents to get out of it.

The second series of experiments was set in the environment with maps randomly generated for every episode.

- A3C-SOM achieved reward 8
- HNS-SOM achieved reward 2.5
- HNS-SOM with additional head achieved reward 19

HNS-SOM and A3C-SOM performed worse, comparing to the 5 fix seeds environment. HNS SOM performed better, achieving better reward.

CONCLUSION

Adding SOM to the agent architecture can help make learning process more robust. It also can increase achieved reward, giving enough complexity to show the difference.

HNS shows high reward in the environment with alternation of fixed maps (seeds).

HNS SOM with additional head achieved maximum possible reward (with respect to steps limitation).