The Malmo Collaborative AI Challenge

Team: bacon-reloaded

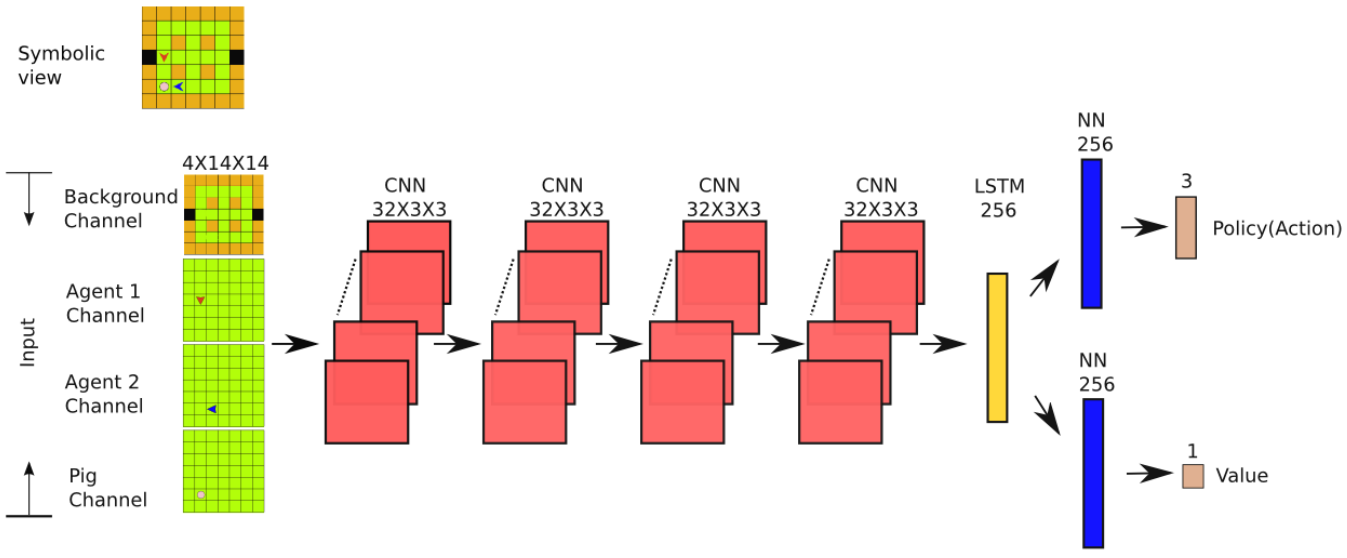
We are **bacon-reloaded** (Jingwei Zhang @jingweiz, Lei Tai @onlytailei, Alexander Schiotka @thalasso007). This is our report for our participation in the **The Malmo Collaborative AI Challenge**.

Summary

Approach

- **Asynchronous Advantage Actor-Critic (A3C) with Generalized Advantage Estimation (GAE)**: We want to train our agent in an end-to-end fashion without any hand-crafted features, so we choose to solve this task using deep reinforcement learning. We implemented A3C [1] along with GAE [2] to train our agent. We choose to use A3C mainly due to the observation that the data is relatively expensive to collect from the malmo environment, thus we want to be able to collect experiences over multiple machines. The asynchronous nature of A3C meets this requirement.
- **Implementation Details**: We use the symbolic view as input to the network, with a little bit of preprocessing: we cut off the borders of the symbolic view since they provide no useful information for the agent, then one symbolic view is fed into the network as a 4-channel image: one for all the objects in the environment, one to indicate the position of the pig, then one for each of the two agents.

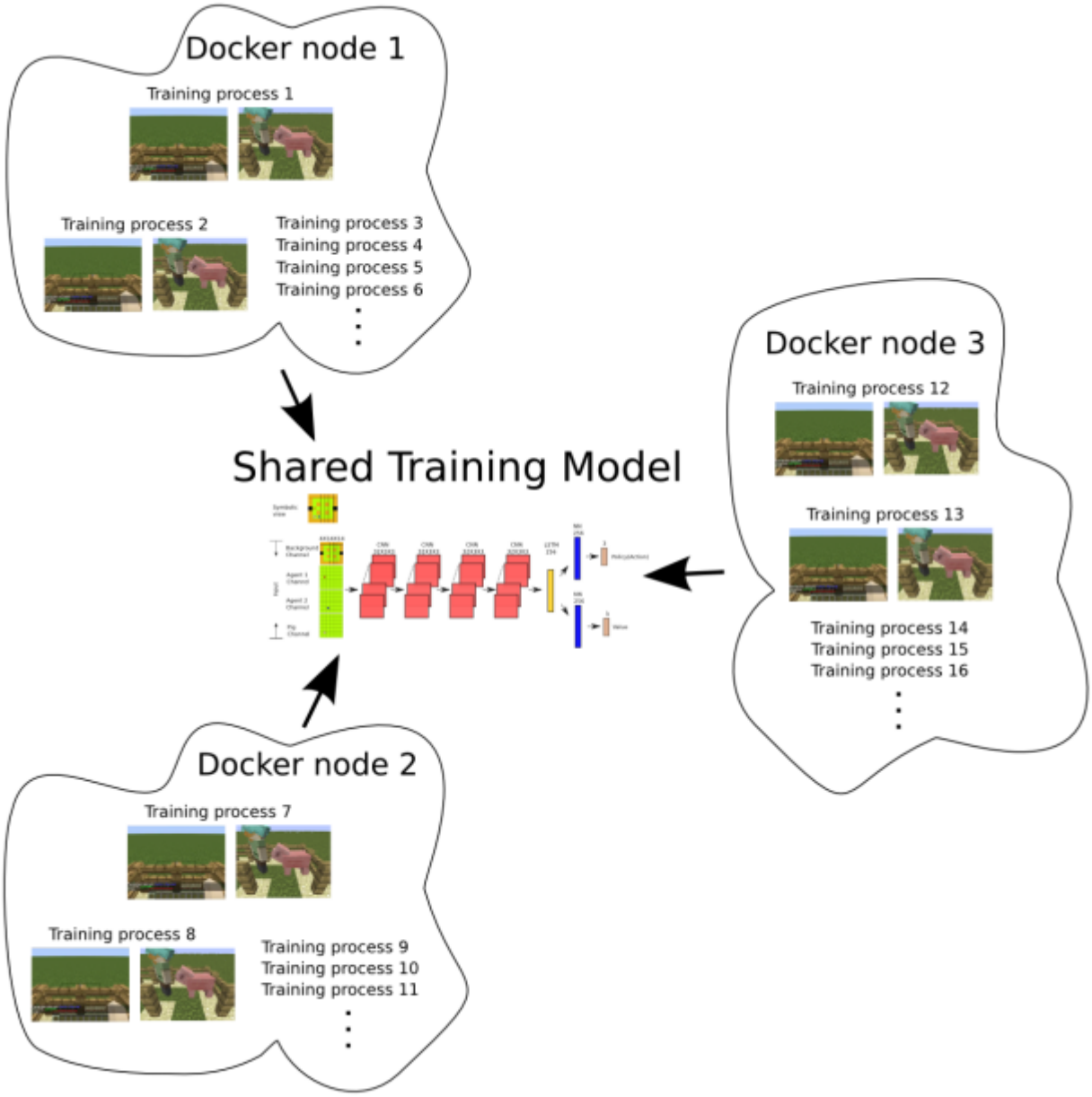
The overall architecture of our network is shown below:



Novel Points

- **Training over multiple machines with docker**: As an asynchronous deep reinforcement learning method, A3C needs to run tens of different training processes at the same time, to break the temporal correlation in the data. With an efficient forward accumulation of rewards, this method shows a progressive result in Atari and OpenAI gym tasks running purely on CPU. As a relatively computational expensive simulation environment, it is impossible to start tens of malmo training processes on one machine (every training process contains two Minecraft environments). Therefore, the swarm mode of docker provides an efficient way to build such a distributed training system, and it can balance the source usage between different swarm nodes. In order to solve the pig-chase task with A3C, we start 16 training processes with 32 Minecraft environments in docker swarm mode and distribute the training over 3 CPU servers. The training process can communicate with all of the Minecraft environments effectively in the docker network.

This distributed training of A3C is shown in the figure below:



*

Curriculum Learning: After training a vanilla A3C agent on the malmo challenge, we found that the agent tend to go to the exit directly to get the small reward instead of trying to find a way to cooperate with its opponent and catch the pig. We suspect that this is due to the probability for the latter scenario to happen is relatively small, so the agent would have too few such examples to learn sufficiently how to get the big reward for catching the pig. This observation inspired us to utilize curriculum learning in the training of our agent, in the belief that it would be much more probable for the agent to learn the optimal behavior if it starts from relatively simple tasks (by simple here we mean that the probability for the agent to catch the pig is higher) then gradually transits to more difficult tasks [3]. We thus modify our training procedure such that in the beginning of the training, the opponent would always be executing A* actions `FocusedAgent`, so the probability that our agent and the opponent could catch the pig together would be higher. The probability that the opponent is a `RandomAgent` instead of a `FocusedAgent` is linearly annealed from `0` to `0.75` (`0.75` is the original setting in the malmo challenge). This gives us a big performance gain even though we have only trained this paradigm for a relatively small number of iterations; we believe this training procedure can result in much better performance if we have trained it for more iterations. * **Periodical Fine-tuning**: One thing we notice is that the malmo environment is relatively unstable when running on remote clusters: we start 16 pig_chase training processes, which contain 32 malmo environments in total, then many of the two environments in the same training process stop to communicate with each other in about 20 minutes. Thus our solution is to restart a new training instance completely every 20 minutes, by fine-tuning from the model saved from the last training instance.

Evaluation Results

Our final evaluation result is here:
https://github.com/jingweiz/malmo-challenge/blob/master/ai_challenge/pig_chase/final_result.json


```
{"500k": {"var": 18.835718608113272, "count": 519, "mean": 0.50481695568400775},
"100k": {"var": 3.4812551696770671, "count": 2158, "mean": -0.85171455050973122},
"experimentname": "My experiment 2"}
```


Our final evaluation result is submitted to the LeaderBoard as **#29** under the Experiment Name as **My experiment 2**.

How to reproduce the result

Training environment configuration

We distribute our training over 3 CPU servers. Firstly we build a docker swarm and let all of those three CPU servers join this swarm. We build a docker swarm by:

In manager node:

```
docker swarm init --advertise-addr <MANAGER node ip>
```

In worker node:

```
docker swarm join \
  --token
SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e
7c \
  <manager node ip>:<port>
```

Create a docker swarm overlay network for the communications between the docker nodes:

```
docker network create --driver overlay --subnet 10.0.9.0/24 --opt encrypted
malmo_net
```

Create a docker volume for models weights saving:

```
docker volume create malmo_volume
```

Build the training docker image: You can find the Dockerfile for `onlytailei:malmo:latest` and `onlytailei/malmopy-pytorch-cpu:latest` in Dockerfile and Dockerfile.

Training

Then in the worker node of docker swarm, start the docker-compose file using docker stack:

```
docker stack deploy --compose-file=docker/malmopy-ai-challenge/docker-compose.yml
malmo_stack
```

The main training code is in *pig_chase_a3c.py*

The weights are saved every 17 mins. They are saved in the created docker volume *malmo_volume*. We can run an arbitrary image with this volume to copy the weights to local.

For periodical fine-tuning, directly run this script:

```
malmo_loop.sh
```

Evaluation

Evaluation is also implemented in docker. The result json will be saved in docker volume *malmo_volume*

```
cd docker/malmopy-eval
docker-compose up
```

Video

We generate a video showing the performance of our agent, which can be found here.

References:

- [1] [Asynchronous Methods for Deep Reinforcement Learning] (https://arxiv.org/abs/1602.01783)
- [2] [High-Dimensional Continuous Control Using Generalized Advantage Estimation] (https://arxiv.org/abs/1506.02438)
- [3] [Curriculum learning] (http://dl.acm.org/citation.cfm?id=1553380)