AI611 Project Report

Team Hackers | Yoonyoung Cho, Minchan Kim, Heesang Jo
AI617: Deep Reinforcement Learning | KAIST SP2022

Introduction

We hypothesize that the NetHack Learning Environment[5] is particularly well-suited for a multi-task learning setup due to the vast diversity of the environment and agent configurations. To validate our hypothesis, the aim of our project was to implement a `Pop-Art`[2] based agent[3] compared to the `IMPALA`[1] baseline agent, which was shown to better address the multi-task learning problem.

Motivation

`A3C` is a distributed actor-critic algorithm that demonstrated excellent performance across many RL domains. However, A3C is a single-task baseline algorithm without any multi-tasking considerations, `IMPALA`[1] extended A3C with the introduction of v-traces to compensate for policy divergence between the behavior policy μ and the learning policy π, with importance-sampling weights, which previously demonstrated compelling performance on multi-tasking ATARI domains.
The results from `IMPALA`[1] were improved further in [3] By employing a `Pop-Art`[2] based task-specific reward-scaling scheme to equalize the influence of each task on the policy, the agent learns to perform multiple tasks simultaneously. This motivates our investigation to try the `Pop-Art` based extension on NLE environment.

Our Architecture

Fig.1 shows the architecture of our feature extractor. We base our implementation – whose output features are shared among the value head and the policy head – on the baseline model from the `NLE` repository. Therefore, our model also has a dedicated `blstats` (bottom-line statistics) encoder which has the same architecture as the baseline and one `LSTM` layer.

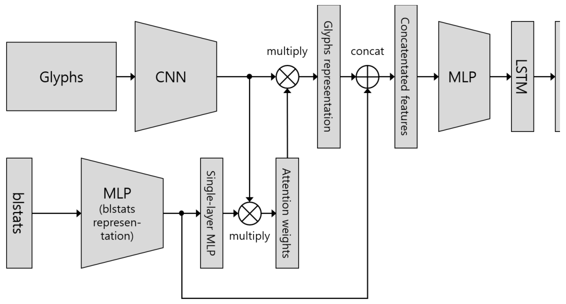

Fig.1. The architecture of our feature extractor. The final CNN layer has a dimension of (64 × 9 × 3). The dimension of the `blstats` embedding is also reduced to 64 by linear projection. We calculate attention weights from the `blstats` embedding (key) and the channels of `CNN` features (query). The resulting weights attend to CNN features (value).

But we also tried to improve it. In particular, we changed how features coming from the glyphs array are computed. The baseline model not only extracts CNN features from the whole glyph array, but also from a 9 × 9 crop of glyphs centered at the agent's current location. The rationale behind this is that this egocentric view is helpful in training the agent.

However, we thought that cropping 9 × 9 glyphs is arbitrary. Instead, we implemented an attention mechanism that takes CNN features from the whole glyphs array as a query and value, and `blstats` embedding as a key. In this way, our agent is not limited to focus on its current location, and it can focus on a wider range of glyphs if needed. We also increased the number of channels in the CNN layer, and used skip connections in ResNet to facilitate gradient flow.

Results

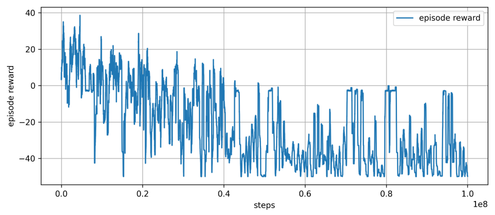

Fig.2. Training progress plot with our agent on `NetHackScore-v0`.

Our initial hypothesis was that `Pop-Art-based` reward scaling scheme would be better suited to solve NetHack, due to the high variability of the procedurally generated environments. However, our experiment clearly shows that our agent's performance was not great.

Fig. 2 shows that our agent fails to converge. In fact, the episode reward decreases to about `-40.0` while training for `0.1B` steps for 9.6 hours on `RTX A6000`. We have been investigating the reason behind this. To see whether it was an issue with the implementation, we validated our agent on standard benchmarks such as `CartPole-v2`. The agent successfully converged, so we know that our implementation is functional.

One observation we made is the behavior of policy collapse, where the agent seems to always output the same action regardless of the change in the environment. This is demonstrated in the probability distribution plot over the 23 actions in NLE, shown in Fig.3:

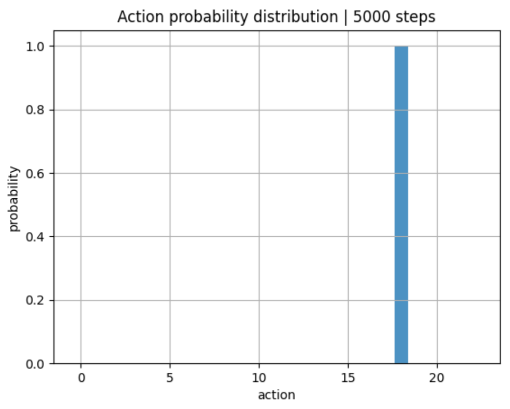

Fig.3. Action distribution plot over 5000 steps.

Indeed, when we check the training process of the agent, it seems to degenerate towards zero gradients, effectively nullifying the training signal. we suspect that one possible cause of this may be the weight decay – if the agent is completely unable to learn from the environment, perhaps the loss is only minimized by collapsing the weight terms.

Another issue that we identified with the current experimental script is that it's not actually running a *multi-task* scenario: we define the `task-id` for pop-art scaling to be dependent on the role, race, alignment, and gender of the character. However, we realized that the environment only runs the `mon-hum-neu-mal` variant by default, when we're running NetHackScore-v0. So, it might not benefit much from the pop-art module.
We're certainly running a lot smaller number of batch sizes and epochs, which could result in poor performance.

Conclusion

We have concentrated on the diversity and multi-tasking nature of NetHack, and proposed to solve this with Pop-Art based IMPALA agent[3] which has shown excellent multi-tasking capability in ATARI domains. However, despite the apparent soundness of our implementation, we found that directly applying Pop-Art-based IMPALA agent was non-trivial.

References

[1] Espeholt, Lasse, et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures." International Conference on Machine Learning. PMLR, 2018.
[2] Hasselt, H. V. et al. “Learning values across many orders of magnitude.” NIPS (2016).
[3] Hessel, Matteo, et al. "Multi-task deep reinforcement learning with popart." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. No. 01. 2019.
[4] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. PMLR, 2016.
[5] Küttler, Heinrich, et al. "The nethack learning environment." Advances in Neural Information Processing Systems 33 (2020): 7671-7684.

Appendix

How to run our Code

Below procedure describes the instructions for running our code:

```
# run in host:
$ ./docker/run.sh
# run in docker image:
$ python3 -m pip install stable-baseline3
tensorboard
$ python3 train_pop_art_agent.py
$ tensorboard --logdir /tmp/nethack
$ python3 record_video.py [CKPT_FILE]
```

Also refer to README.md from our repository.