DRLearner

Open Source Deep Reinforcement Learning
AGI-22
Chris Poulin

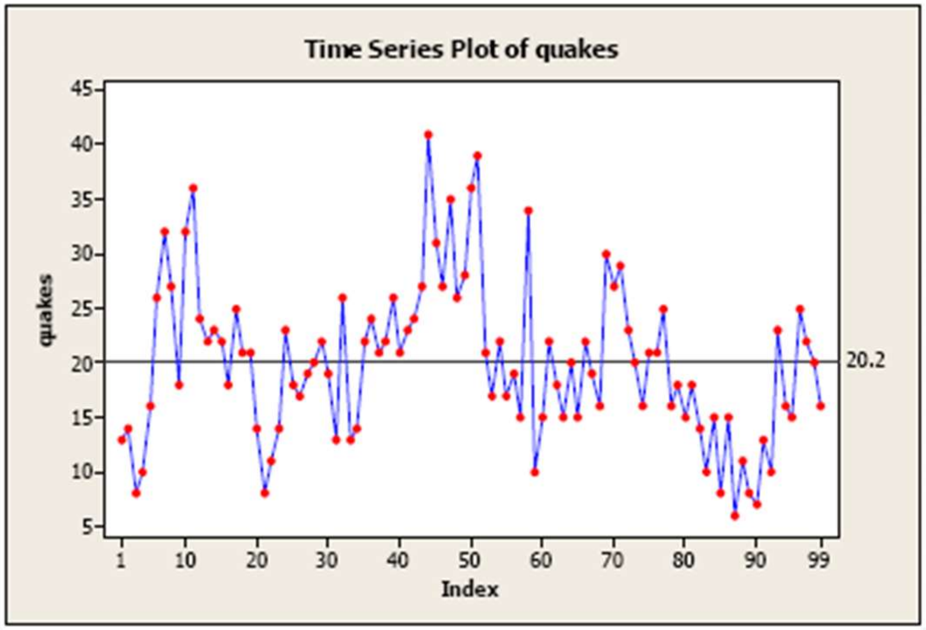
DRLearner.org

Time Series, RL, and Temporal Difference Learning

Time Series, and advanced analysis of time-based decisions

- A time series is a series of data points in time order. Most commonly [1].
  A time series is usually a sequence (equally spaced points in time) [2]
- Reinforcement learning (RL) is an area of machine learning concerned with how agents take actions in an environment to maximize reward [3][4]
- Temporal difference (TD) learning refers to model-free reinforcement learning methods which learn by bootstrapping from the current value function.[5] Was discovered that the firing rate of dopamine neurons appear to mimic the error function in the algorithm. [4]
- And many people were working intensely on the math around these problems (e.g. Schmidhuber et al) [6]

- [1] https://en.wikipedia.org/wiki/Time_series
  [2] https://online.stat.psu.edu/stat510/book/export/html/661
  [3] https://en.wikipedia.org/wiki/Reinforcement_learning
  [4] Richard Sutton & Andrew Barto (1998). Reinforcement Learning. MIT Press. ISBN 978-0-585-02445-5
  [5] ttps://en.wikipedia.org/wiki/Temporal_difference_learning
  [6] https://people.idsia.ch/~juergen/naturedeepmind.html



Source: "Time Series Basics", Penn State [2]

DeepMind’s Breakthrough

However, there were a high-profile series of papers that…

- Demonstrated human-level game play on Atari games
- *“Playing Atari with Deep Reinforcement Learning”* (arXiv, 2013) [7]
- *“Human-level control through deep reinforcement learning”* (Nature, 2015) [8]
- This “deep Q network” started a frenzy in machine Learning around the area of “Deep Reinforcement Learning” or “DRL”
- [7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. Playing Atari with Deep Reinforcement Learning. Tech Report, 19 Dec. 2013, http://arxiv.org/abs/1312.5602

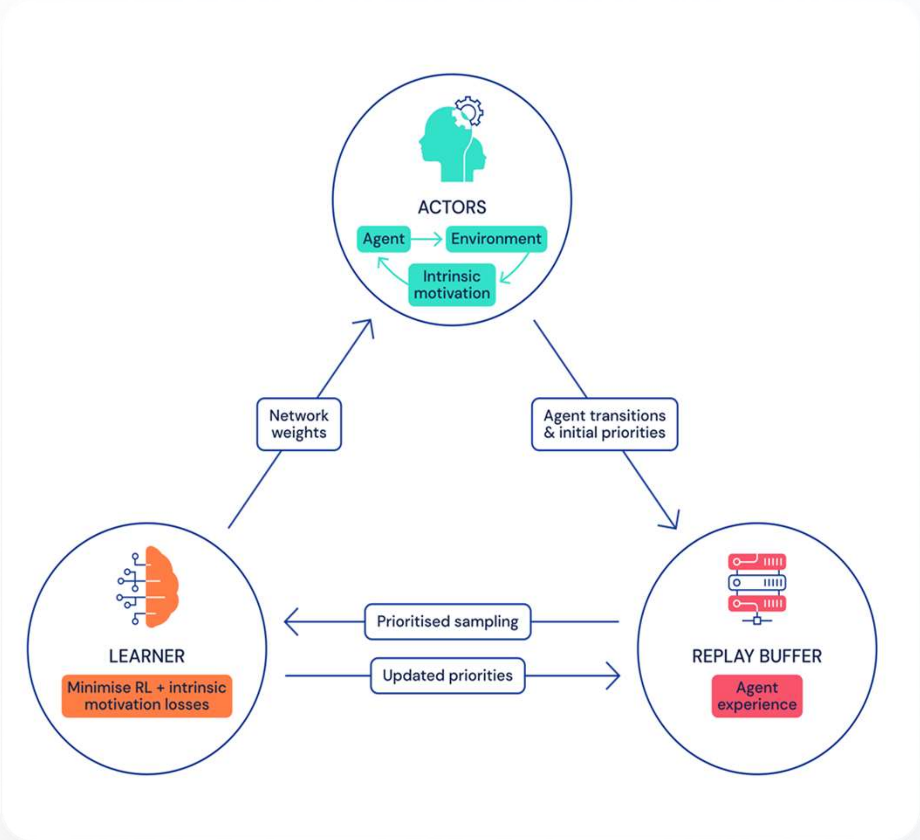
  [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis. Human-level control through deep reinforcement learning. Nature, vol. 518, p 1529, 26 Feb. 2015. http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html

Source: DeepMind

Agent 57 (from DeepMind)

[Agent57: Outperforming the Atari human benchmark](#)
By Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell - 2020

- **Actors**, which include Environments and Agents
- * The Agents in this case, have a different scoring paradigm (intrinsic motivation) in **CURIOSITY**
- A **Learner**, which is a Reinforcement Learning front end to a (any) Neural Network.
- Using a **Replay Buffer** (R2D2) to give the system episodic memories to pull from.
- **Massively parallel**, requiring thousands of GPU hours



Source: Agent 57, Badia et. al.

Agent 57: Performance

Agent 57 shows better than human performance

- **Mastery level game play** (compared to human average)
- Compared to state-of-the-art ML (2020**), best performance by some metrics** (capped mean)
- Only **second place in overall average for DeepMind's "best"** (2020)
- Relatively **simple architecture**, given the high performance.
- Implementation is still relatively new (non-public)

Statistics	Agent57	NGU	R2D2 (Retrace)	R2D2	MuZero
Capped mean	**100.00**	95.07	94.20	94.33	89.92
Number of games $>$ human	**57**	51	52	52	51
Mean	4766.25	3421.80	3518.36	4622.09	**5661.84**
Median	1933.49	1359.78	1457.63	1935.86	**2381.51**
40th Percentile	1091.07	610.44	817.77	1176.05	1172.90
30th Percentile	614.65	267.10	420.67	529.23	503.05
20th Percentile	**324.78**	226.43	267.25	215.31	171.39
10th Percentile	**184.35**	107.78	116.03	115.33	75.74
5th Percentile	**116.67**	64.10	48.32	50.27	0.03

Source: Agent 57, Badia et. al.

Agent 57: Demo

[Here Agent 57 is playing Atari Games]

- The Arcade Learning Environment (ALE) is a standard tool for time dependent agents to compare performance
- **Zaxxon** is a "medium" difficulty game
- The classic **PitFall** gets into "hard" category. As it requires a large amount of understanding of the 'game map'.
- **Montezuma's Revenge** is considered one of the "hardest" Atari games, in terms of game space complexity.
- *In all three cases, Agent 57 is better than an average human player.*

Zaxxon
https://www.youtube.com/watch?v=FhJ2yLzi4Kk&list=PL2D_SqpHWZGgQu4gIUARUWk3rrwBrBEgq

Pitfall
https://www.youtube.com/watch?v=96UNx8SvU_U

Montezuma's Revenge
https://www.youtube.com/watch?v=A32KP0DCbaE

Source: Agent 57, Badia et. al.

Our journey: LIT Review

1 We wanted to duplicate the capability of DeepMind's Agent 57, announced 2020, here: **Agent 57: Outperforming the human Atari benchmark**

https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

2 This was apparently an extension of prior work (Badia et al) on a system called NGU, here: **Never Give Up: Learning Directed Exploration Strategies**

https://arxiv.org/abs/2002.06038

3 Once we started communication with Adria Badia, he recommended that I start my efforts based on another effort called R2D2, here: **Recurrent Experience Replay in Distributed Reinforcement Learning (R2D2)**

https://deepmind.com/research/publications/2019/recurrent-experience-replay-distributed-reinforcement-learning

4 Adria and Chris then discussed various implementations of this available online, and we thought that the best baseline would be the ACME framework from DeepMind, here: **Acme: A Research Framework for Distributed Reinforcement Learning**

https://arxiv.org/abs/2006.00979
https://github.com/deepmind/acme

5 Experimented with ACME, and was able to successfully implement/test (on GCP) most of these examples, including R2D2, here:

https://github.com/deepmind/acme/tree/master/examples

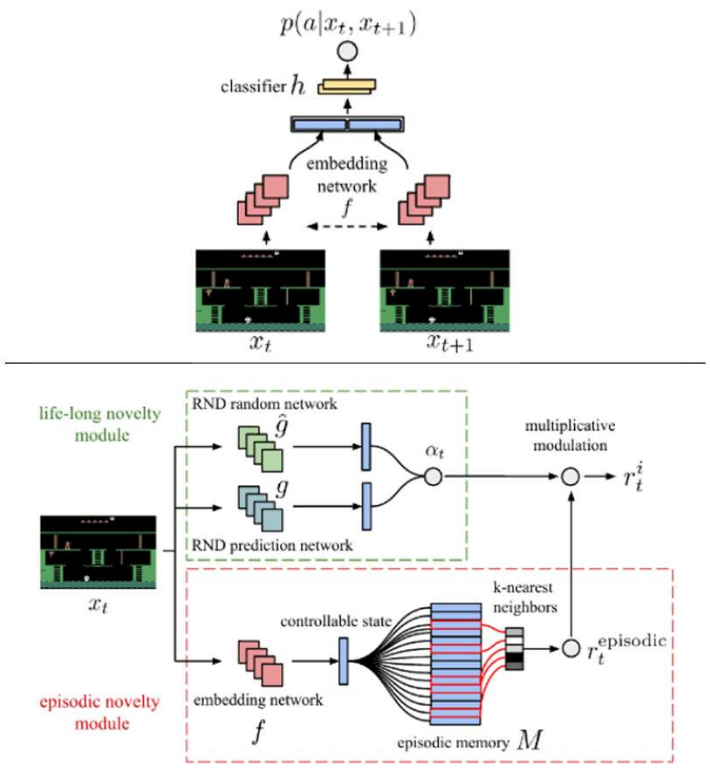
6 Where we left off was that Adria recommended combining the functionality of NGU with the functionality of R2D2 (on Acme), as he thought this would be a lesser clone of Agent 57 that he could make further suggestions on. He has provided a few code samples, but we (Chris) got stuck on the TF matrix transformations.

acme

Never Give Up (NGU)

NGU IS A BOREDOM FREE CURIOSITY-BASED RL METHOD

- Converts screen images to abstract feature representations
- Features used to learn controllable states and to generate intrinsic rewards
- Needed a modification that was applied to open-source ACME R2D2
- DeepMind tools used to design embedding networks, episodic novelty module, and calculate intrinsic rewards
- Single threaded and lacking the life-long novelty module



Credit: Never Give Up: Learning Directed Exploration Strategies, Badia et. al.

NGU: Implementation

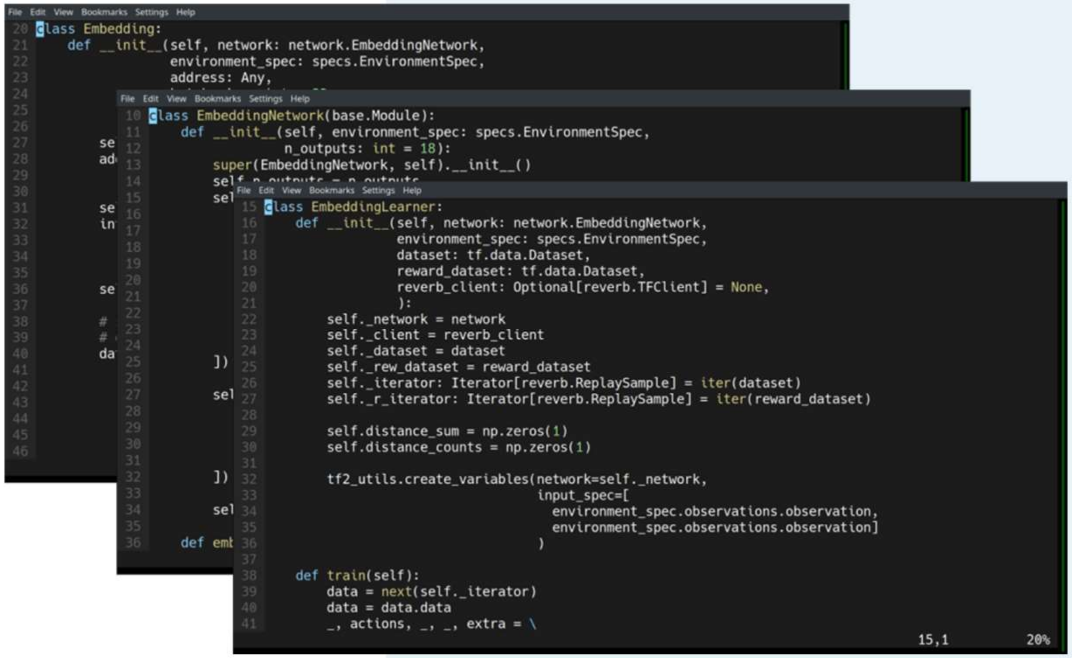
With advice from Badia et al, we first created:

- Implementation of episodic memory using reverb data table extra spec

- Curiosity rewards implemented using k-nearest neighbors in embedding space

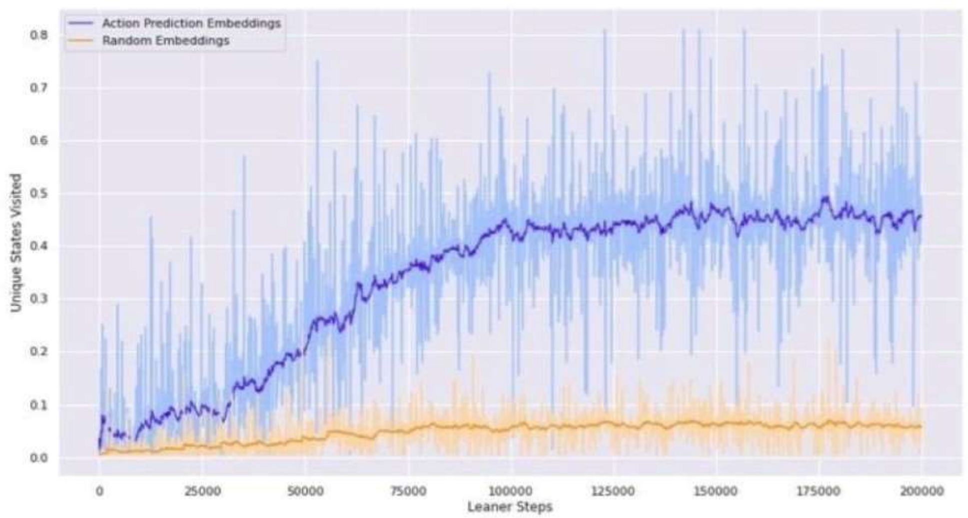
- Implement embedding network architecture using sonnet

- Store state embedding representations in reverb data table



TensorFlow to JAX

- Switched from TensorFlow to JAX as backend framework to enable *distributed implementation*
- Inverse dynamics model for intrinsic rewards computations – verified by DiscoMaze experiment
- Random Network Distillation for intrinsic reward modulation
- Replace n-step bootstrapping with retrace learning algorithm
- Replace Q-network with Universal Value Function Approximator (UVFA)



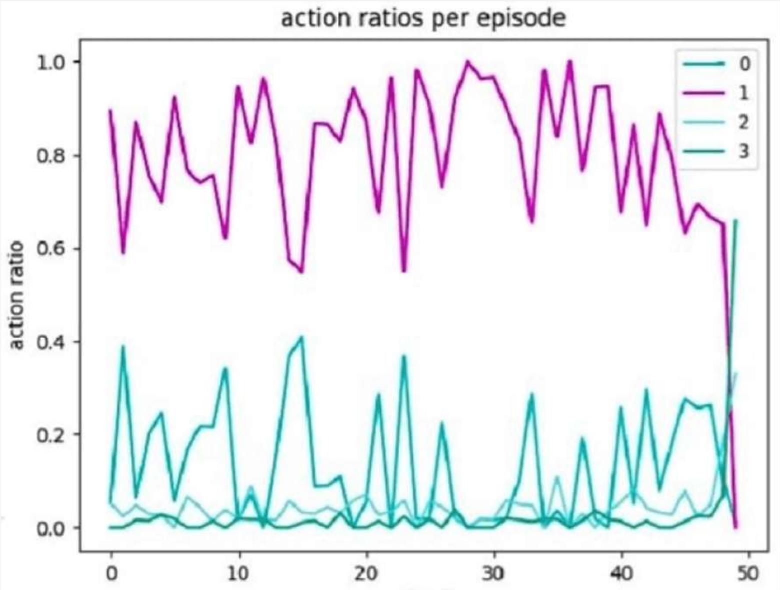
Initial Testing

Restoring model from checkpoint

- Implemented script for restoring model from checkpoint, because it wasn't initially supported by ACME
- Note, that optimizer state and training metadata are preserved, so training can be continued at any point

Metrics customization

- Implemented flexible logging module which can be extended with any custom metric
- Added new metric (actions ratio per episode), which was not previously available in the framework



Actions ratios per episode plot

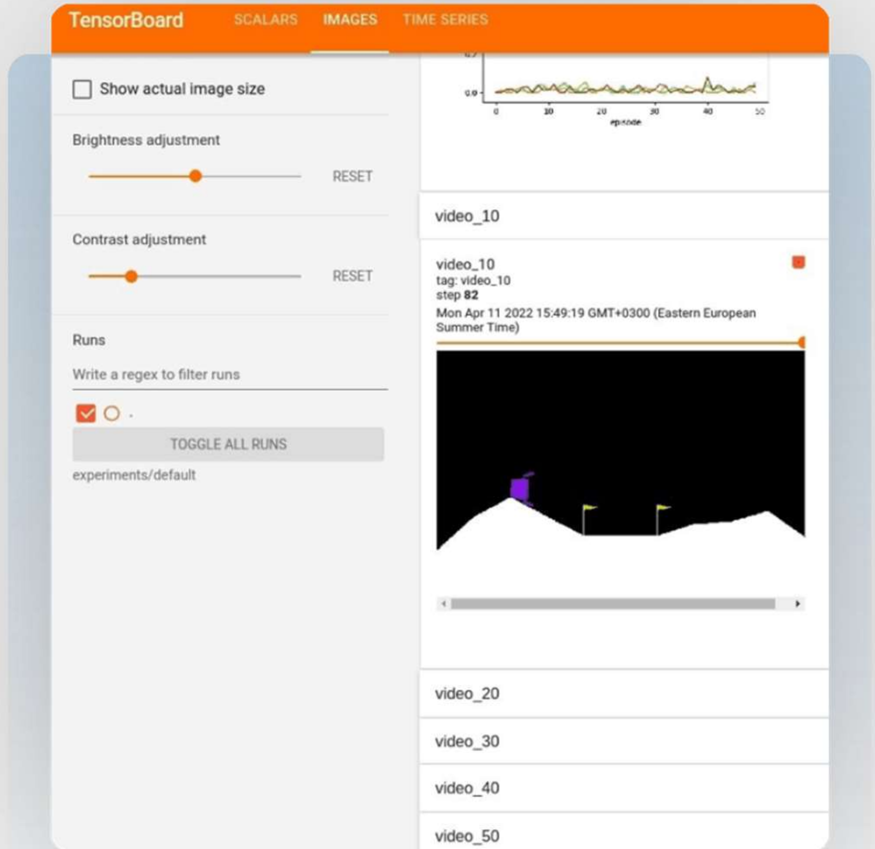
Testing and Logging

LOG EPISODE EXAMPLES

Also, saving episode examples was implemented for monitoring model performance over time.

Episode examples are logged each n-th episode

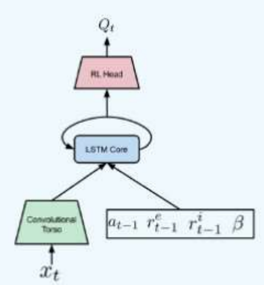
Videos can be logged into TensorBoard or locally



Lunar Lander episode stored in TensorBoard

Rewards Modeling

Separate parametrization for extrinsic (points) and intrinsic (curiosity) rewards



(a) Sketch of the R2D2 Agent

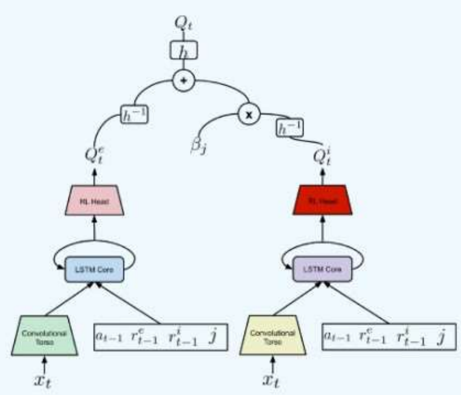
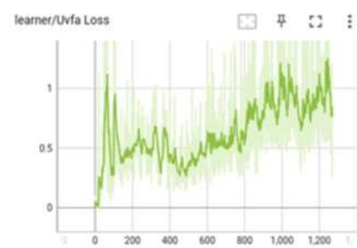
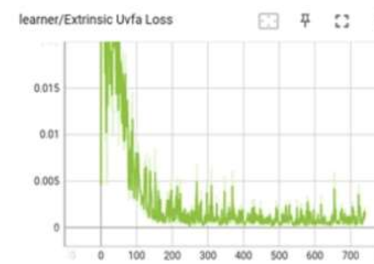
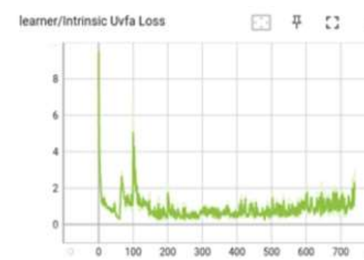


Figure 9. Sketch of the Agent57.

The implementation of the split of UVFA training mostly includes changes in learner, and the inference in the actor.



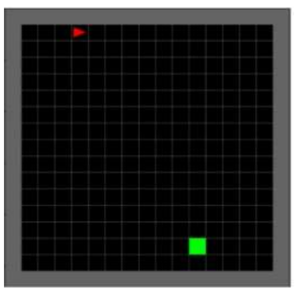




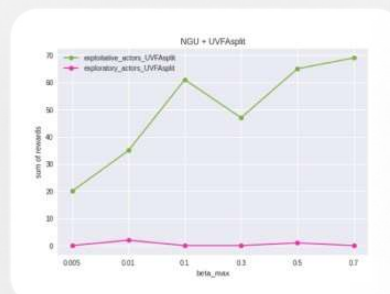
Experiments testing Rewards

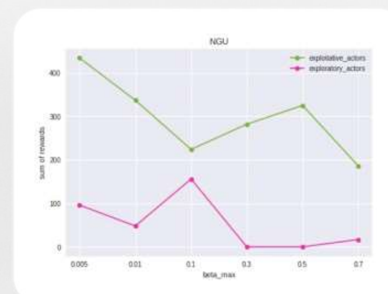
Random coin environment

- 4 actions: move right, left, up, down. Maximum 200 moves.
- an agent(a red arrow) and a coin(a green square) are randomly placed in 15x15 grid
- when agent steps over a coin a reward of 1 is given and the episode terminates
- Gym-minigrid environment here: Random Coin



- Further comparison shows that indeed NGU is unstable as the network tries to learn both exploratory and exploitative policies jointly.
- The bigger the beta_max parameter the more sum of returns for exploitative NGU agents decreases on random coin environment as opposed to the split version





META Controller/Meta-learning

- **Selects which policy to use at training and evaluation time**
- Policies are represented by [β, γ] pair (also called mixtures – larger mixture index means more exploratory behavior)
- Results in agent learning *when it's better to explore and when to exploit*
- Implemented as UCB multi-armed bandit
- Extrinsic episode returns are used as rewards for the bandit
- Each actor has its own meta controller
- Implemented in actor_core.py

Fully Distributed Training

1

Distributed agents can be trained in multiple processes or on multiple machines.

2

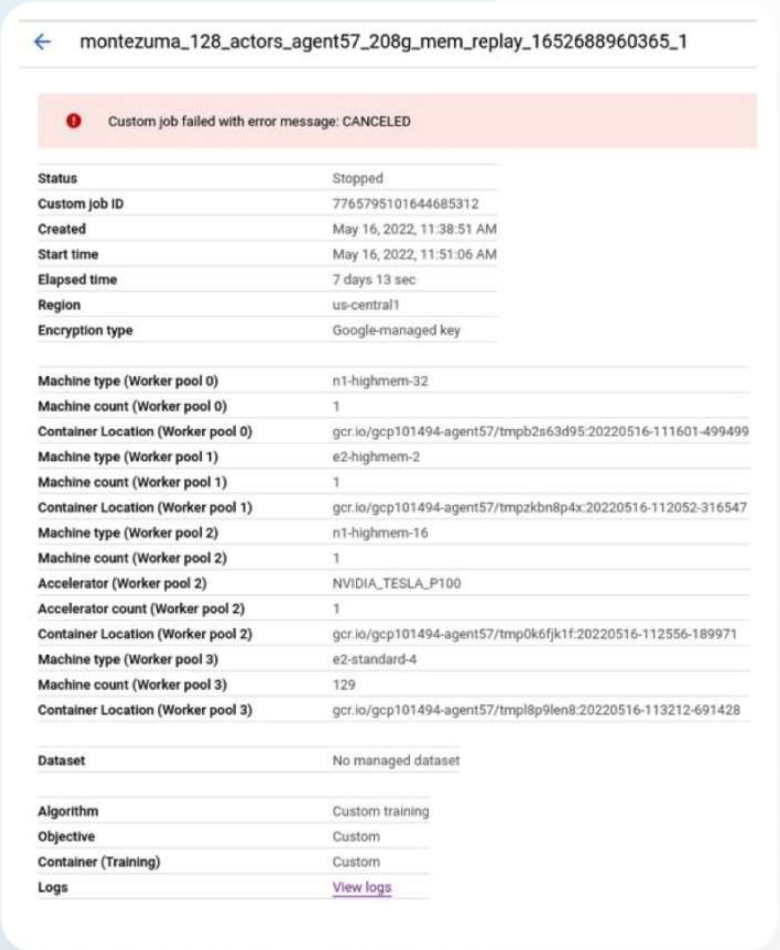
Multiple machines training is executed on Vertex AI – a GCP for building and deploying AI models.

3

In a distributed setup, communication between nodes is handled by launchpad package.

VERTEX AI TRAINING PIPELINE:

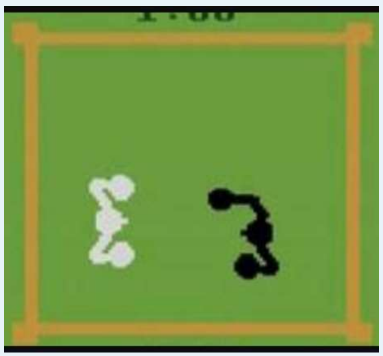
- Package the code for every type of launchpad node (Actor, Learner, Replay Buffer) into a Docker container (using launchpad)
- Build the Docker images locally or on Cloud Build
- Specify the hardware requirements for each node type
- Create a custom job on Vertex AI to train the agent
- All training artifacts are saved into a Cloud Storage bucket.



Screenshot of Vertex AI training job configuration

Game Environment Validation

Arcade Learning Environment (ALE)



Easy: Boxing

Medium: Zaxxon

Hard: Montezuma Revenge

Other Modality Testing: Procgen

Coinrun

Code

Source Code:
https://github.com/PatternsandPredictions/DRLearner_beta

Dev Mailing List:
https://groups.google.com/g/drlearner/

Contact:

Chris Poulin, Project Lead
chris@patternsandpredictions.com

Sponsored by Patterns and Predictions
patternsandpredictions.com

Lessons Learned & Next Steps

- Computational cost (cloud) is non-trivial
- Noise reduction needed (e.g. contrastive learning?)
- Refocus on Representations-AGI
- Continuous environments (e.g. robotics) combinatorially explosive
- API Exploration (other modalities)
- Better documenting/on boarding (e.g. instructional videos)
- Come join us!

DRLearner

Thank you

Chris Poulin (Project Lead-US)
Phil Tabor (Co-Lead-US)
Dzvinka Yarish (Ukraine)
Ostap Viniavskyi (Ukraine)
Oleksandr Buiko (Ukraine)
Yuriy Pryyma (Ukraine)
Mariana Temnyk (Ukraine)
Volodymyr Karpiv (Ukraine)
Mykola Maksymenko (Advisor-Ukraine)
Iurii Milovanov (Advisor-Ukraine)

DRLearner.org