DRLearner

Open Source Reinforcement Learning: Deep Dive

AGI-22

Phil Tabor

DRLearner.org

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:

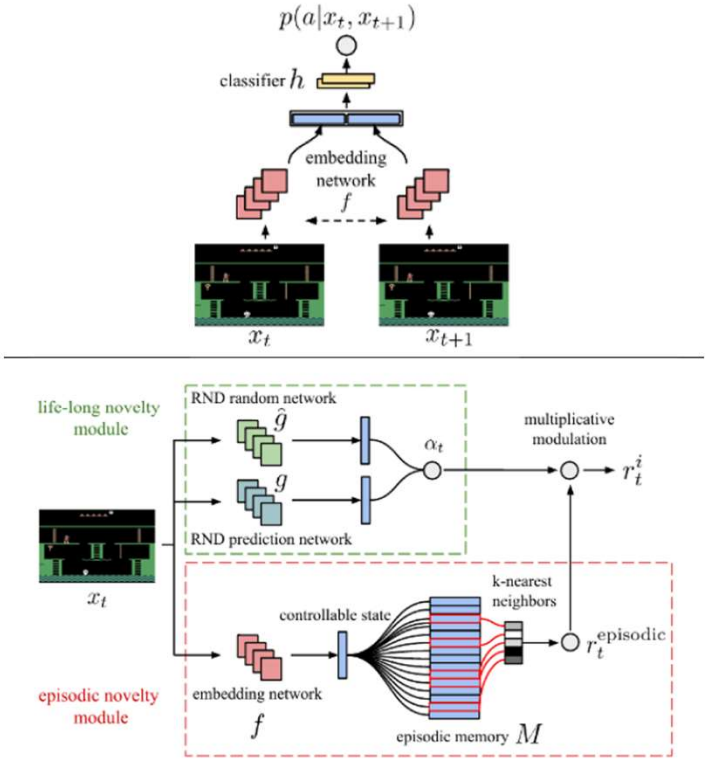
   https://github.com/deepmind/acme/tree/master/examples

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.

Curiosity Learning and NGU

NGU IS A BOREDOM FREE CURIOSITY-BASED RL METHOD

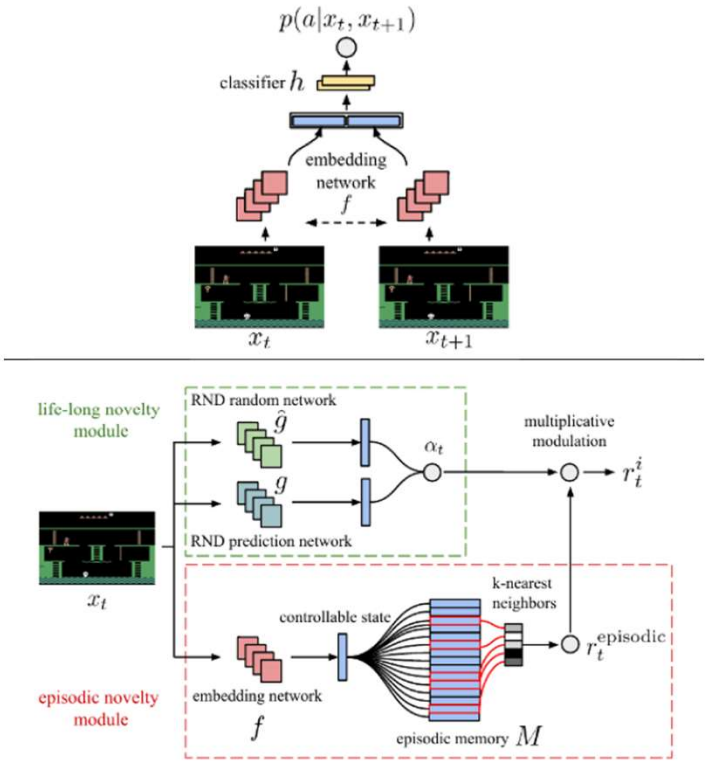
- Curiosity: learning in environments with sparse rewards
- Give intrinsic rewards to the agent based on its inability to predict actions generating successive states
- As the agent visits the same (or similar states) it gets better at predicting transitions, thus intrinsic rewards go to 0 (boredom)
- Another pitfall is rewarding passive observation (agent is rewarded for observing unpredictable noise without acting)
- Never Give Up solves these problems using concepts of lifelong/episodic curiosity and controllable states



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

NGU Architecture

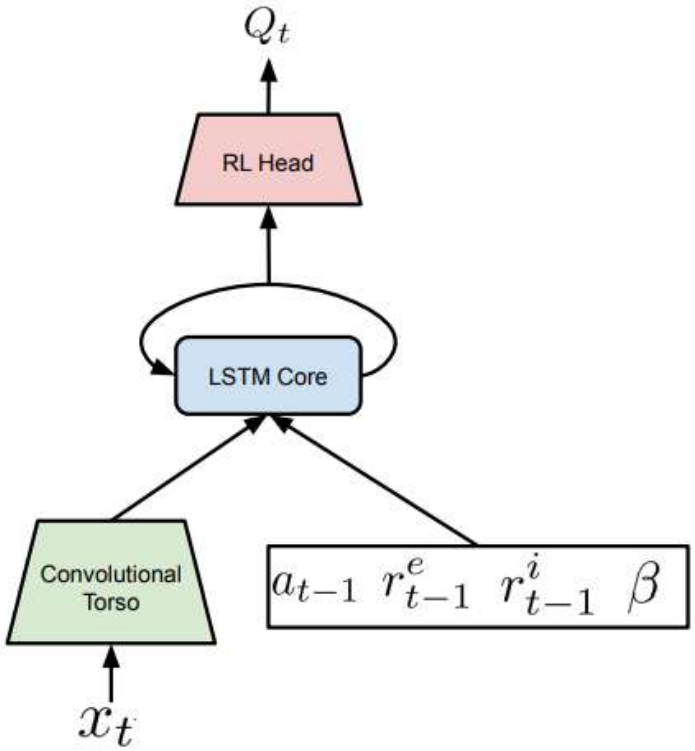
- CNN converts screen images to abstract feature representations
- Embedding features used to learn controllable states and to generate intrinsic rewards
- Distance between inter-episode controllable states is used to generate intrinsic rewards (episodic curiosity)
- RND network generates multiplicative constant for the episodic reward (life-long curiosity)
- What about the explore-exploit dilemma?



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

NGU Architecture

- Use UVFA to approximate $Q(x, a, \theta, \beta_i) \rightarrow r_t = r^e + \beta_i r^i$
- Discrete number of β between β_{min} and β_{max} where we include 0 and 1.
- Turn off exploratory policy by acting greedily with respect to $Q(x, a, \theta, 0)$
- Learn to exploit without seeing any extrinsic reward
- Concatenate one hot encoding of beta to action and both rewards and feed into LSTM core for the agent

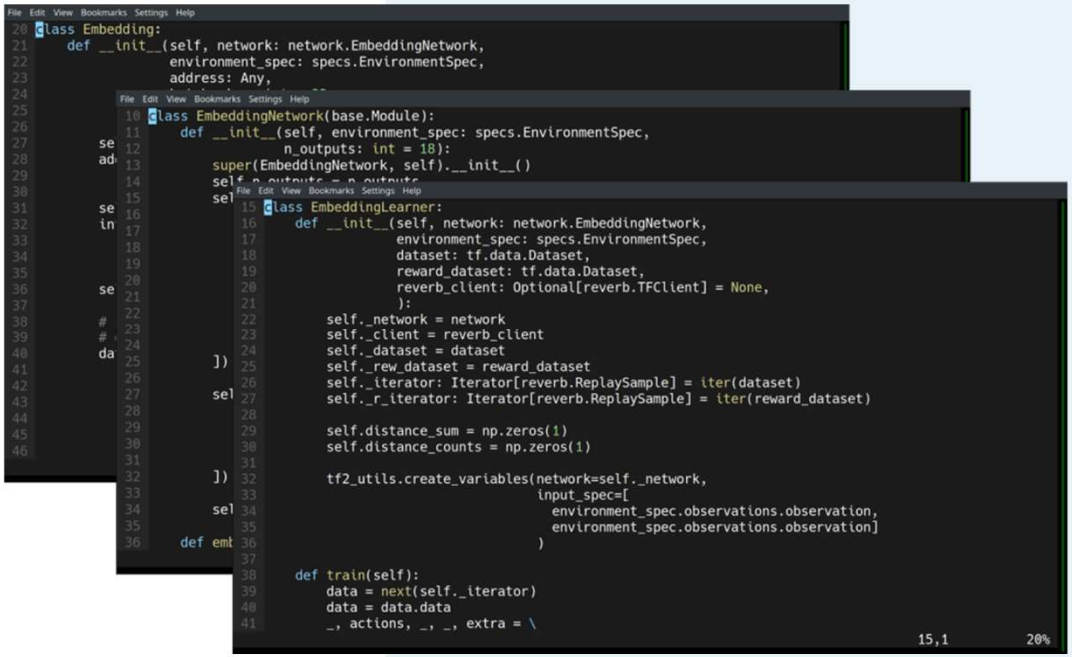


Credit: Never Give Up: Learning Directed Exploration Strategies, Badia et. al.

NGU Implementation

V0.1 Implementation

- Start with minimum viable implementation of never give up
- Use DeepMind's tools (ACME) to the greatest extent possible
- Sonnet for the embedding network
- Reverb for data storage and retrieval



```
class Embedding:
    def __init__(self, network: network.EmbeddingNetwork,
                 environment_spec: specs.EnvironmentSpec,
                 address: Any,
```

```
class EmbeddingNetwork(base.Module):
    def __init__(self, environment_spec: specs.EnvironmentSpec,
                 n_outputs: int = 18):
        super(EmbeddingNetwork, self).__init__()
```

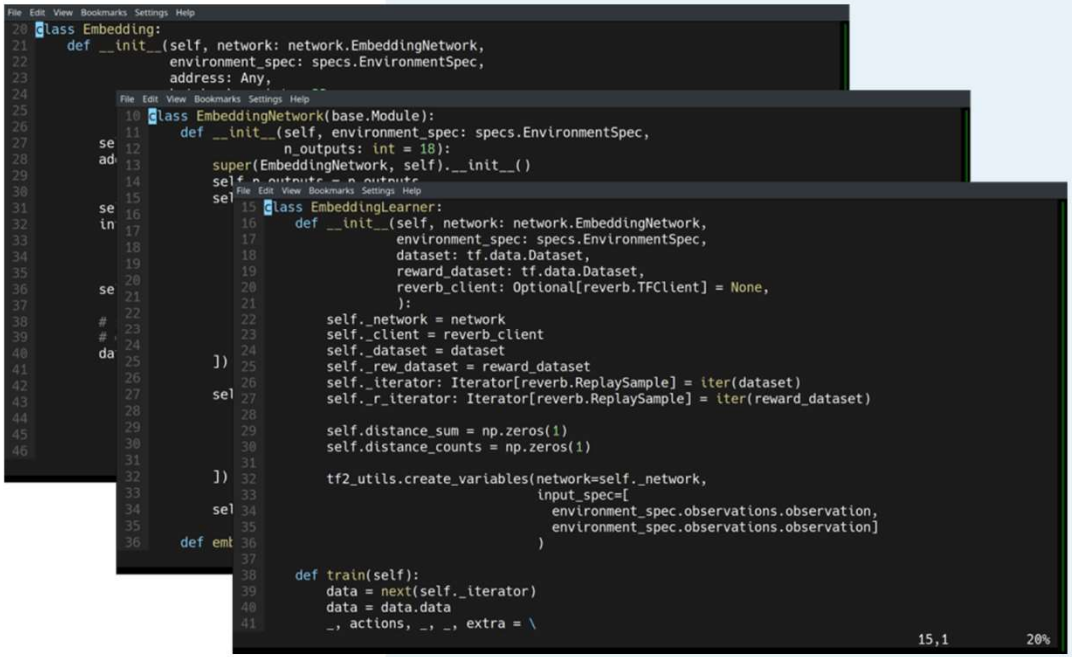
```
class EmbeddingLearner:
    def __init__(self, network: network.EmbeddingNetwork,
                 environment_spec: specs.EnvironmentSpec,
                 dataset: tf.data.Dataset,
                 reward_dataset: tf.data.Dataset,
                 reverb_client: Optional[reverb.TFClient] = None,
                 ):
        self._network = network
        self._client = reverb_client
        self._dataset = dataset
        self._rew_dataset = reward_dataset
        self._iterator: Iterator[reverb.ReplaySample] = iter(dataset)
        self._r_iterator: Iterator[reverb.ReplaySample] = iter(reward_dataset)

        self.distance_sum = np.zeros(1)
        self.distance_counts = np.zeros(1)

        tf2_utils.create_variables(network=self._network,
                                   input_spec=[
                                     environment_spec.observations.observation,
                                     environment_spec.observations.observation]
                                   )

    def train(self):
        data = next(self._iterator)
        data = data.data
        _, actions, _, _, extra = \
```

15,1 20%

NGU Implementation

V0.1 Implementation Drawbacks

- Lack of multithreading (TF2)
- No life-long curiosity module
- Used the n-step loss instead of retrace loss
- Didn’t use UVFA, so no β



```
class Embedding:
    def __init__(self, network: network.EmbeddingNetwork,
                 environment_spec: specs.EnvironmentSpec,
                 address: Any,
```

```
class EmbeddingNetwork(base.Module):
    def __init__(self, environment_spec: specs.EnvironmentSpec,
                 n_outputs: int = 18):
        super(EmbeddingNetwork, self).__init__()
```

```
class EmbeddingLearner:
    def __init__(self, network: network.EmbeddingNetwork,
                 environment_spec: specs.EnvironmentSpec,
                 dataset: tf.data.Dataset,
                 reward_dataset: tf.data.Dataset,
                 reverb_client: Optional[reverb.TFClient] = None,
                 ):
        self._network = network
        self._client = reverb_client
        self._dataset = dataset
        self._rew_dataset = reward_dataset
        self._iterator: Iterator[reverb.ReplaySample] = iter(dataset)
        self._r_iterator: Iterator[reverb.ReplaySample] = iter(reward_dataset)

        self.distance_sum = np.zeros(1)
        self.distance_counts = np.zeros(1)

        tf2_utils.create_variables(network=self._network,
                                   input_spec=[
                                     environment_spec.observations.observation,
                                     environment_spec.observations.observation]
                                   )

    def train(self):
        data = next(self._iterator)
        data = data.data
        _, actions, _, _, extra = \
```

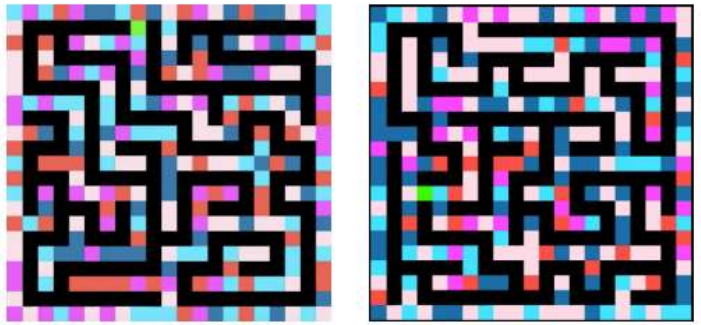
From R2D2 to Full NGU

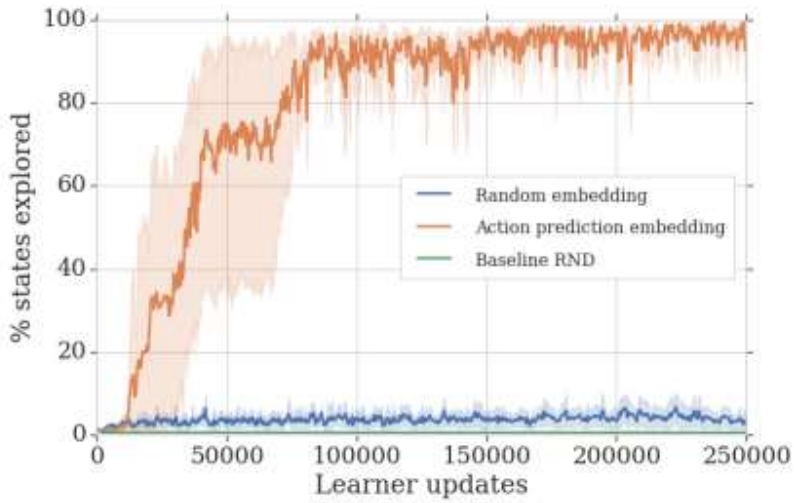
- Switched from TensorFlow to JAX as backend framework to enable distributed implementation
- Inverse dynamics model for intrinsic rewards computations – verified by DiscoMaze experiment
- Random Network Distillation for intrinsic reward modulation
- Replace n-step bootstrapping with retrace learning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

```python
"""Defines local DRLearner agent, using JAX."""

from typing import Optional

from acme import specs
from acme.utils import counting

from ..core import local_layout
from .builder import DRLearnerBuilder
from .config import DRLearnerConfig
from .networks import make_policy_networks, DRLearnerNetworks


class DRLearner(local_layout.LocalLayout):
    """Local agent for DRLearner.

    This implements a single-process DRLearner agent.
    """

    def __init__(
            self,
            spec: specs.EnvironmentSpec,
            networks: DRLearnerNetworks,
            config: DRLearnerConfig,
            seed: int,
            workdir: Optional[str] = '~/acme',
            counter: Optional[counting.Counter] = None,
    ):
        ngu_builder = DRLearnerBuilder(networks, config, num_actors_per_mixture=1)
        super().__init__(
            seed=seed,
            environment_spec=spec,
            builder=ngu_builder,
            networks=networks,
            policy_network=make_policy_networks(networks, config),
            workdir=workdir,
            min_replay_size=config.min_replay_size,
            samples_per_insert=config.samples_per_insert if config.samples_per_insert \
                else 10 / (config.burn_in_length + config.trace_length),
            batch_size=config.batch_size,
            num_sgd_steps_per_step=config.num_sgd_steps_per_step,
            counter=counter,
        )
```

Disco Maze Test

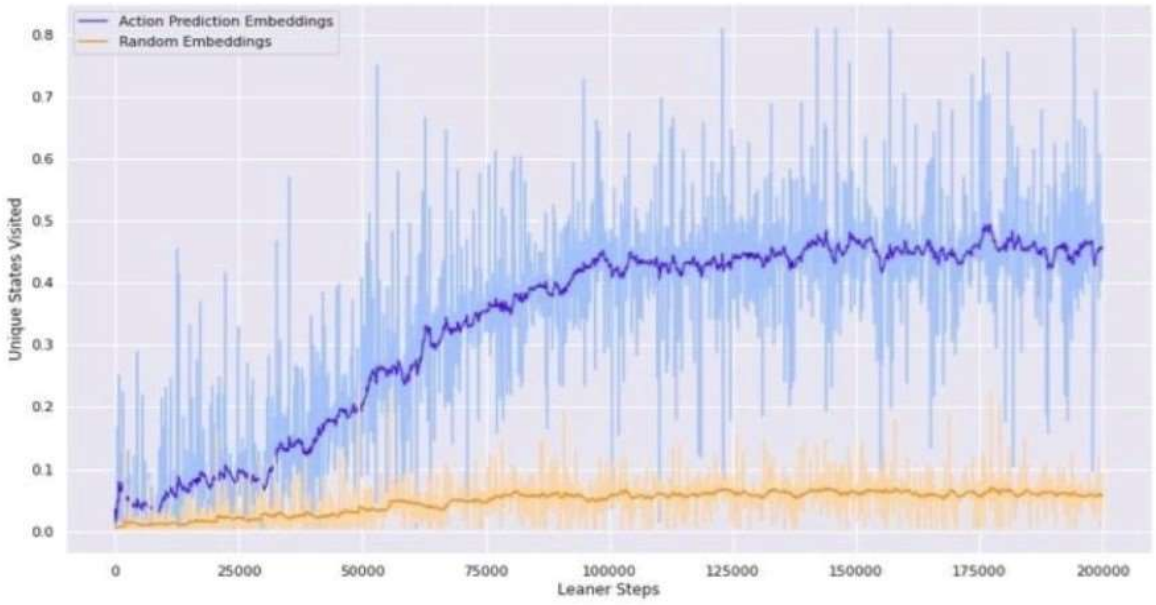
- Test single actor implementation in disco maze environment
- 21x21 randomly generated grid; no extrinsic rewards
- At each time step the blocks change color, which tempts pathological behavior
- NGU agent manages to explore a significant fraction of the available states
- RND and random embeddings flop in this environment





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

Our Disco Maze Results



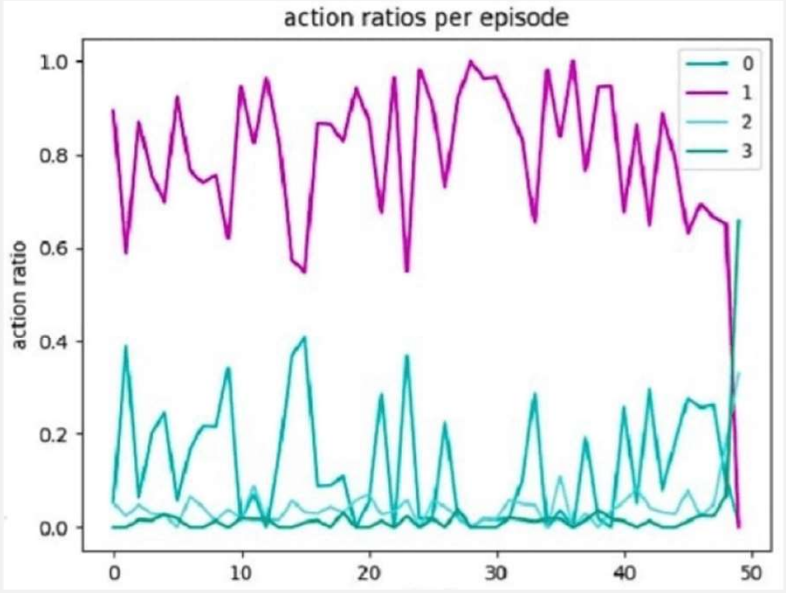
Expanding the ACME Feature Set

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



Action ratios per episode plot

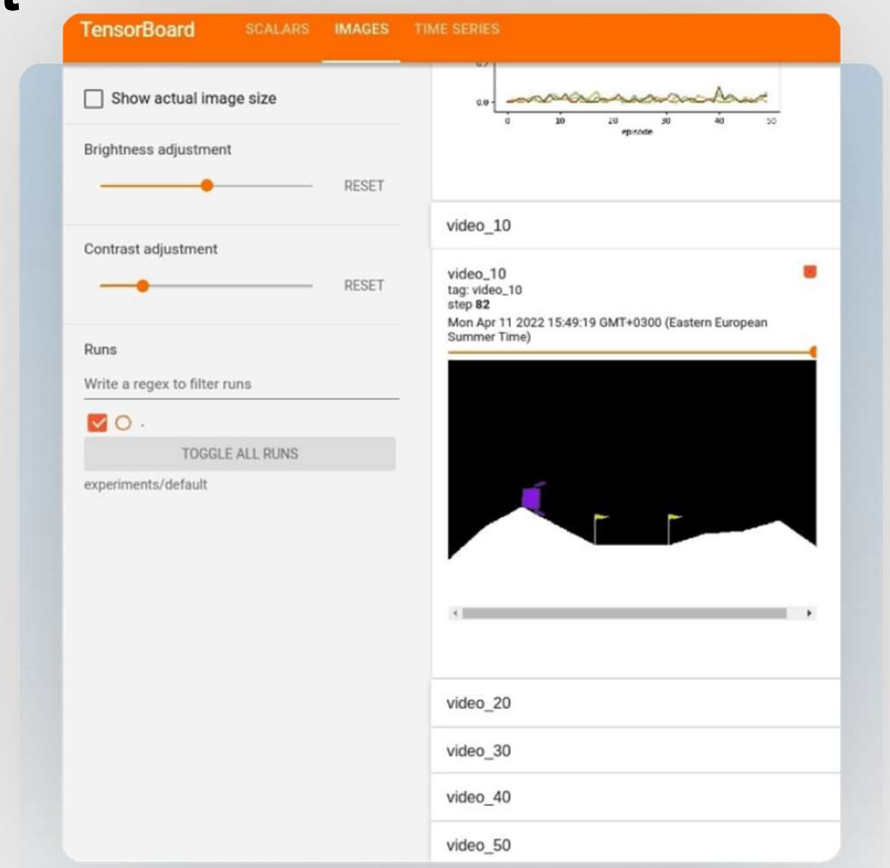
Expanding the ACME Feature Set

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

Beyond NGU

- DeepMind eventually improved upon NGU with Agent57

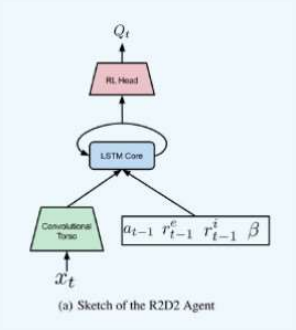
- Split parameterization for UVFA
  $Q(x, a, \theta, j) = Q(x, a, \theta^e, j) + \beta_j Q(x, a, \theta^i, j)$

- Meta-controller for automatically selecting which policy to use

```python
class DRLearnerLearner(acme.Learner):
    """DRLearnerlearner."""

    def __init__(self,
                 uvfa_unroll: networks_lib.FeedForwardNetwork,
                 uvfa_initial_state: networks_lib.FeedForwardNetwork,
                 idm_action_pred: networks_lib.FeedForwardNetwork,
                 distillation_embed: networks_lib.FeedForwardNetwork,
                 batch_size: int,
                 beta_min: float,
                 beta_max: float,
                 gamma_min: float,
                 gamma_max: float,
                 num_mixtures: int,
                 random_key: networks_lib.PRNGKey,
                 burn_in_length: int,
                 target_epsilon: float,
                 importance_sampling_exponent: float,
                 max_priority_weight: float,
                 target_update_period: int,
                 iterator: Iterator[reverb.ReplaySample],
                 uvfa_optimizer: optax.GradientTransformation,
                 idm_optimizer: optax.GradientTransformation,
                 distillation_optimizer: optax.GradientTransformation,
                 idm_clip_steps: int,
                 distillation_clip_steps: int,
                 retrace_lambda: float,
                 tx_pair: rlax.TxPair = rlax.SIGNED_HYPERBOLIC_PAIR,
                 clip_rewards: bool = False,
                 max_abs_reward: float = 1.,
                 use_core_state: bool = True,
                 prefetch_size: int = 2,
                 replay_client: Optional[reverb.Client] = None,
                 counter: Optional[counting.Counter] = None,
                 logger: Optional[loggers.Logger] = None):
        """Initializes the learner."""

        batched_epsilon_greedy_prob = jax.vmap(
            jax.vmap(epsilon_greedy_prob, in_axes=(0, None), out_axes=0),
            in_axes=(0, None),
            out_axes=0
        )

        def uvfa_loss(
                uvfa_params: networks_lib.Params,
                uvfa_target_params: networks_lib.Params,
                key_grad: networks_lib.PRNGKey,
                sample: reverb.ReplaySample,
                rewards_t: jnp.ndarray,
                core_state_extraction_name: str = 'extrinsic_core_state'
        ) -> Tuple[jnp.ndarray, jnp.ndarray]:
            """
            Computes mean transformed N-step loss for a batch of sequences.
            """
```

UVFA Parametrization

Separate parametrization for extrinsic and intrinsic rewards



(a) Sketch of the R2D2 Agent

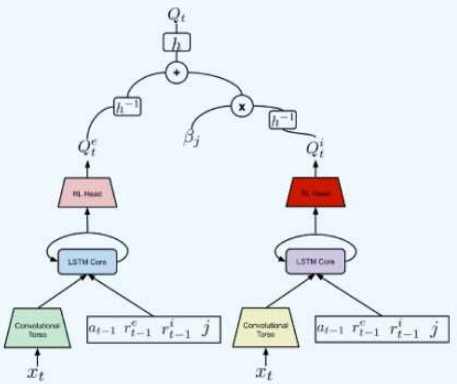
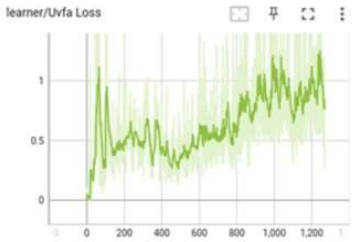
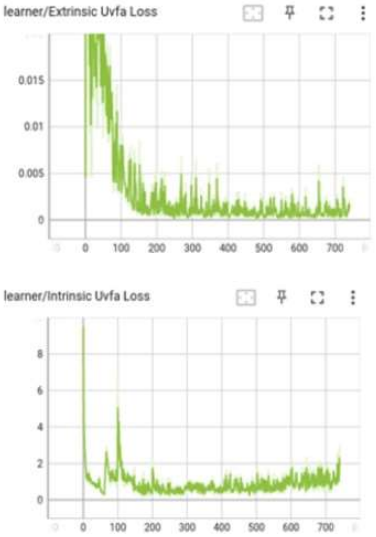


Figure 9. Sketch of the Agent57.

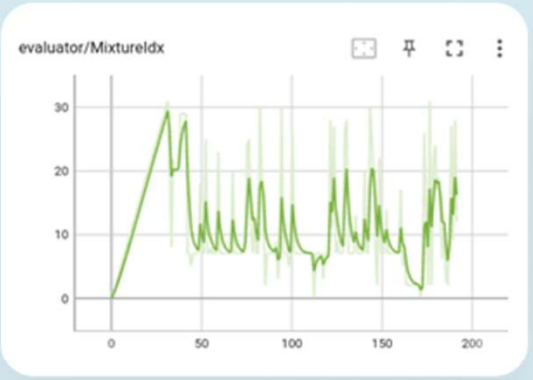
The implementation of the split in UVFA training mostly includes changes in learner, and the inference in the actor.

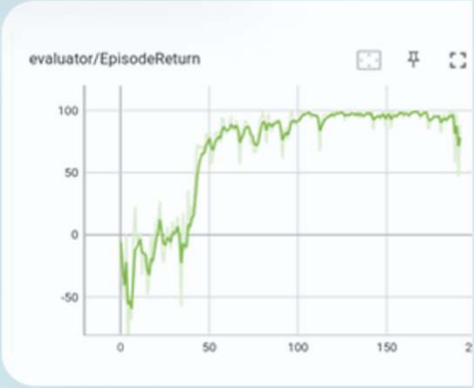




META Controller

- 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





Distributed Training

1

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

2

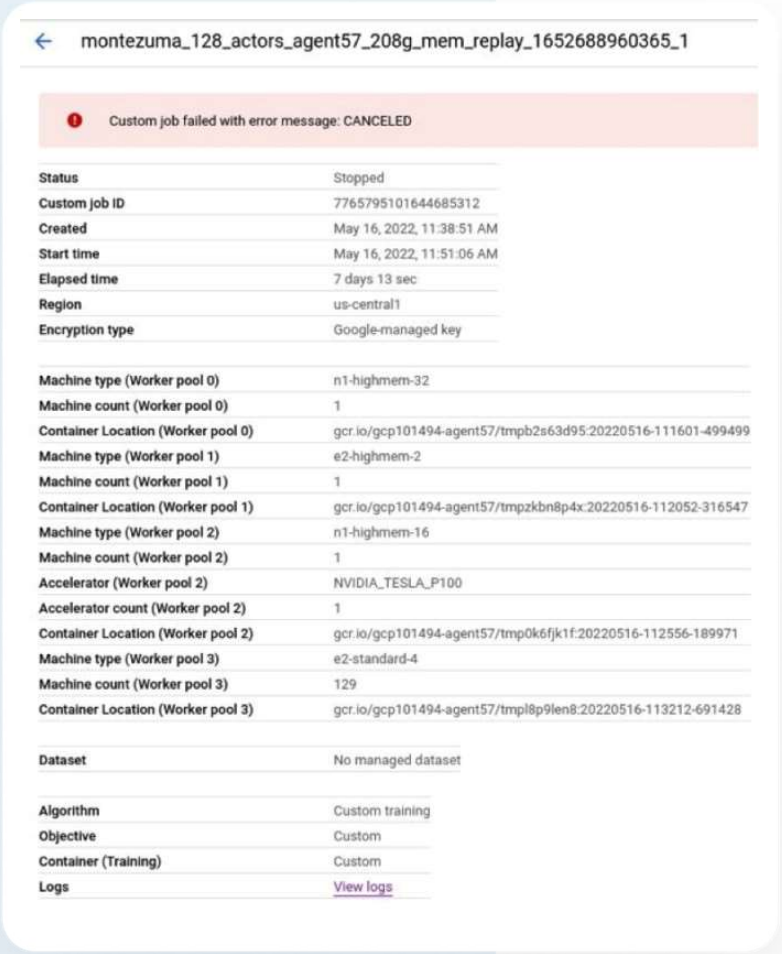
Multiple machines training is executed on Vertex AI – a GCP for building and deploying AI models.

3

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

VERTEX AI TRAINING PIPELINE:

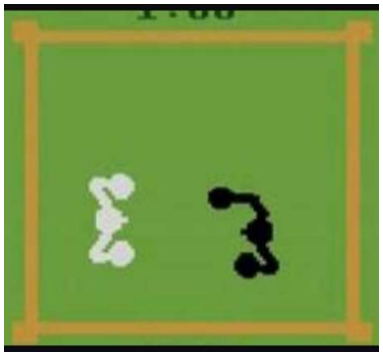
- Package the code for every type of launchpad node (Actor, Learner, Replay Buffer) into a Docker container (using launchpad)
- Build the Docker images locally or on Cloud Build
- Specify the hardware requirements for each node type
- Create a custom job on Vertex AI to train the agent
- All training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



Screenshot of Vertex AI training job configuration

Training Environments

Arcade Learning Environment (ALE)



Easy: Boxing

Medium: Zaxxon

Hard: Montezuma Revenge

Code

Apache License

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

Contact:

Chris Poulin, Project Lead
chris@patternsandpredictions.com

Sponsored by Patterns and Predictions
www.patternsandpredictions.com