V-trace, PopArt Normalization, Partially Observable MDPs

Milan Straka

January 7, 2019

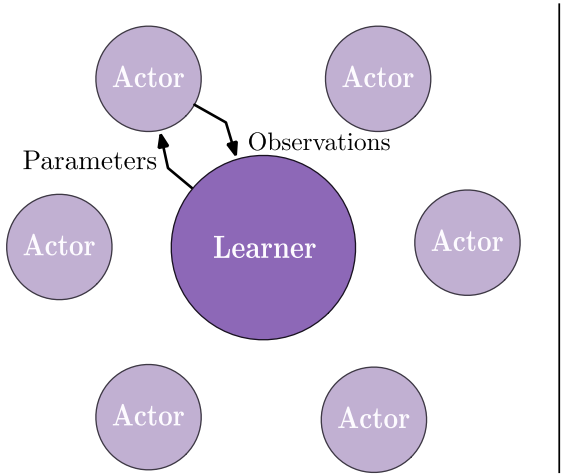
EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

IMPALA

Impala (**Imp**ortance Weighted **A**ctor-**L**earner **A**rchitecture) was suggested in Feb 2018 paper and allows massively distributed implementation of an actor-critic-like learning algorithm.

Compared to A3C-based agents, which communicates gradients with respect to the parameters of the policy, IMPALA actors communicates trajectories to the centralized learner.



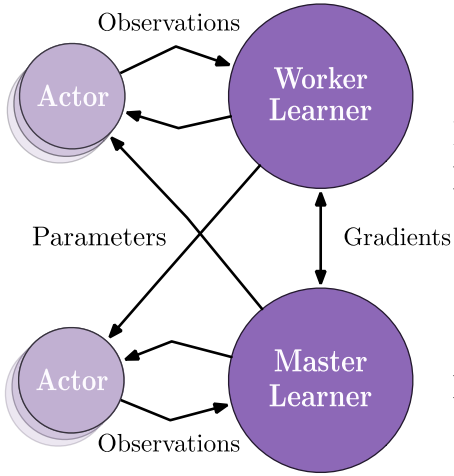


Figure 1 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

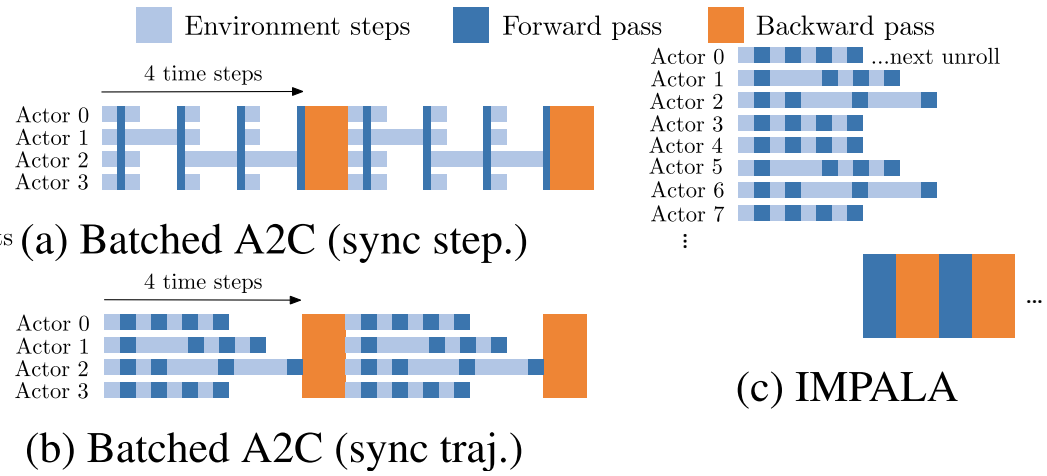


Figure 2 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

If many actors are used, the policy used to generate a trajectory can lag behind the latest policy. Therefore, a new **V-trace** off-policy actor-critic algorithm is proposed.

IMPALA – V-trace

Consider a trajectory $(S_t, A_t, R_{t+1})_{t=s}^{t=s+n}$ generated by a behaviour policy b.

The n-step V-trace target for S_s is defined as

$$v_s \overset{\text{def}}{=} V(S_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left(\prod_{i=s}^{t-1} c_i \right) \delta_t V,$$

where $\delta_t V$ is the temporal difference for V

$$\delta_t V \overset{\text{def}}{=} \rho_t \big(R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \big),$$

and ρ_t and c_i are truncated importance sampling ratios with $\bar{\rho} \geq \bar{c}$:

$$\rho_t \overset{\text{def}}{=} \min \left(\bar{\rho}, \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \right), \quad c_i \overset{\text{def}}{=} \min \left(\bar{c}, \frac{\pi(A_i|S_i)}{b(A_i|S_i)} \right).$$

Note that if $b = \pi$ and assuming $\bar{c} \geq 1$, v_s reduces to n-step Bellman target.

IMPALA – V-trace

Note that the truncated IS weights ρ_t and c_i play different roles:

- The ρ_t appears in the definition of $\delta_t V$ and defines the fixed point of the update rule. For $\bar{\rho} = \infty$, the target is the value function v_π, if $\bar{\rho} < \infty$, the fixed point is somewhere between v_π and v_b. Notice that we do not compute a product of these ρ_t coefficients.
- The c_i impacts the speed of convergence (the contraction rate of the Bellman operator), not the sought policy. Because a product of the c_i ratios is computed, it plays an important role in variance reduction.

The paper utilizes $\bar{c} = 1$ and out of $\bar{\rho} \in \{1, 10, 100\}$, $\bar{\rho} = 1$ works empirically the best.

IMPALA – V-trace

Consider a parametrized functions computing $v(s;\boldsymbol{\theta})$ and $\pi(a|s;\boldsymbol{\omega})$. Assuming the defined n-step V-trace target

$$v_s \stackrel{\text{def}}{=} V(S_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left(\prod_{i=s}^{t-1} c_i \right) \delta_t V,$$

we update the critic in the direction of

$$\big(v_s - v(S_s;\boldsymbol{\theta})\big) \nabla_{\boldsymbol{\theta}} v(S_s;\boldsymbol{\theta})$$

and the actor in the direction of the policy gradient

$$\rho_s \nabla_{\boldsymbol{\omega}} \log \pi(A_s|S_s;\boldsymbol{\omega}) \big(R_{s+1} + \gamma v_{s+1} - v(S_s;\boldsymbol{\theta})\big).$$

Finally, we again add the entropy regularization term $H(\pi(\cdot|S_s;\boldsymbol{\theta}))$ to the loss function.

Architecture	CPUs	GPUs[1]	FPS[2]	
Single-Machine			Task 1	Task 2
A3C 32 workers	64	0	6.5K	9K
Batched A2C (sync step)	48	0	9K	5K
Batched A2C (sync step)	48	1	13K	5.5K
Batched A2C (sync traj.)	48	0	16K	17.5K
Batched A2C (dyn. batch)	48	1	16K	13K
IMPALA 48 actors	48	0	17K	20.5K
IMPALA (dyn. batch) 48 actors[3]	48	1	21K	24K
Distributed				
A3C	200	0	46K	50K
IMPALA	150	1	80K	
IMPALA (optimised)	375	1	200K	
IMPALA (optimised) batch 128	500	1	250K	

[1] Nvidia P100 [2] In frames/sec (4 times the agent steps due to action repeat). [3] Limited by amount of rendering possible on a single machine.

Table 1 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

IMPALA – Population Based Training

For Atari experiments, population based training with a population of 24 agents is used to adapt entropy regularization, learning rate, RMSProp ε and the global gradient norm clipping threshold.

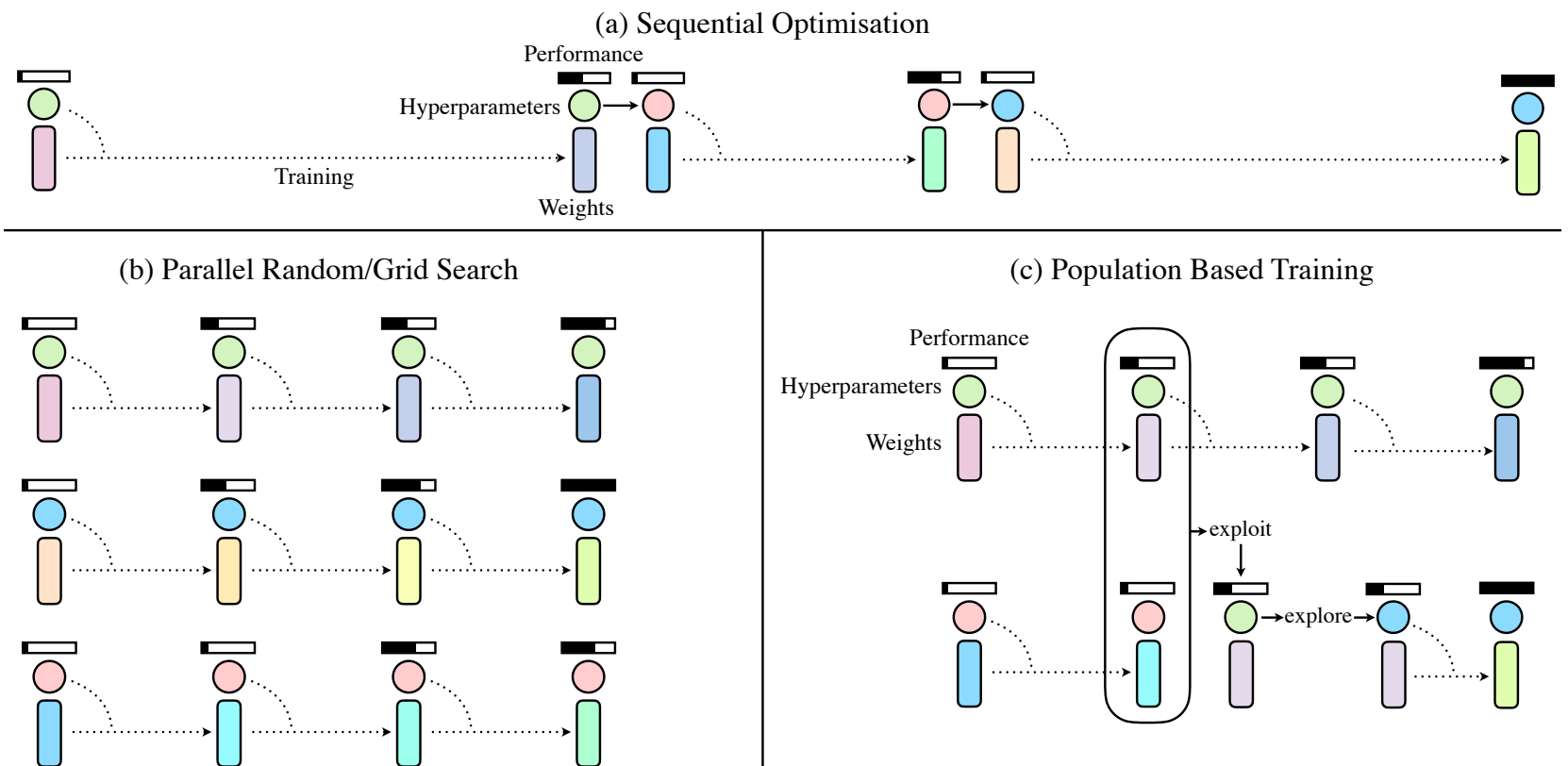


Figure 1 of paper "Population Based Training of Neural Networks" by Max Jaderberg et al.

IMPALA – Population Based Training

For Atari experiments, population based training with a population of 24 agents is used to adapt entropy regularization, learning rate, RMSProp ε and the global gradient norm clipping threshold.

In population based training, several agents are trained in parallel. When an agent is *ready* (after 5000 episodes), then:

- it may be overwritten by parameters and hyperparameters of another agent, if it is sufficiently better (5000 episode mean capped human normalized score returns are 5% better);
- and independently, the hyperparameters may undergo a change (multiplied by either 1.2 or 1/1.2 with 33% chance).

IMPALA

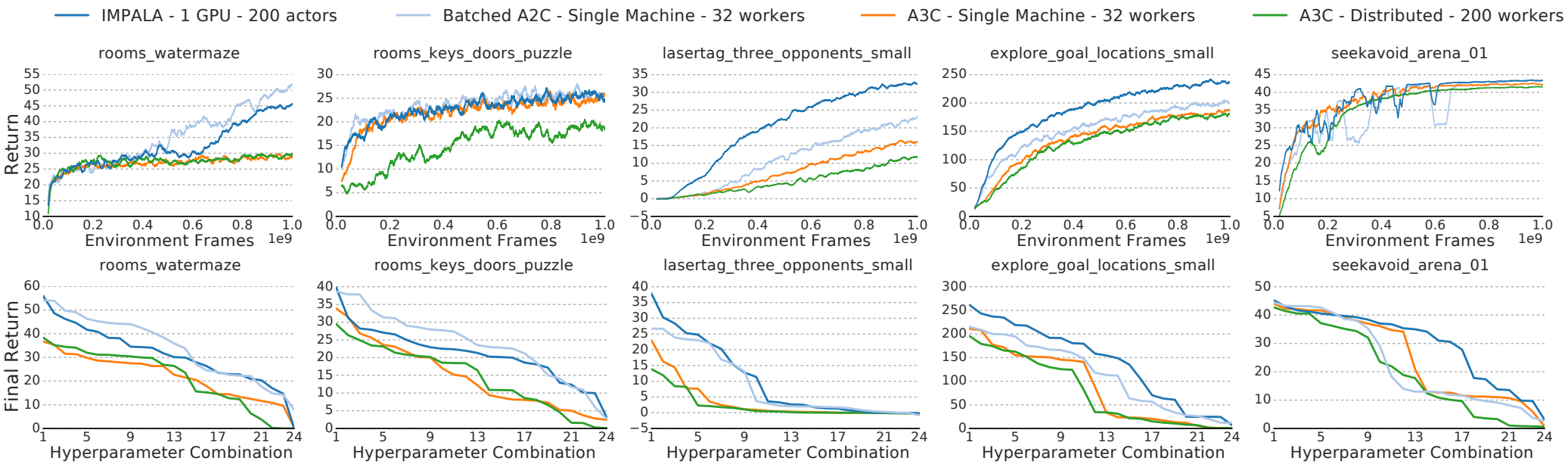
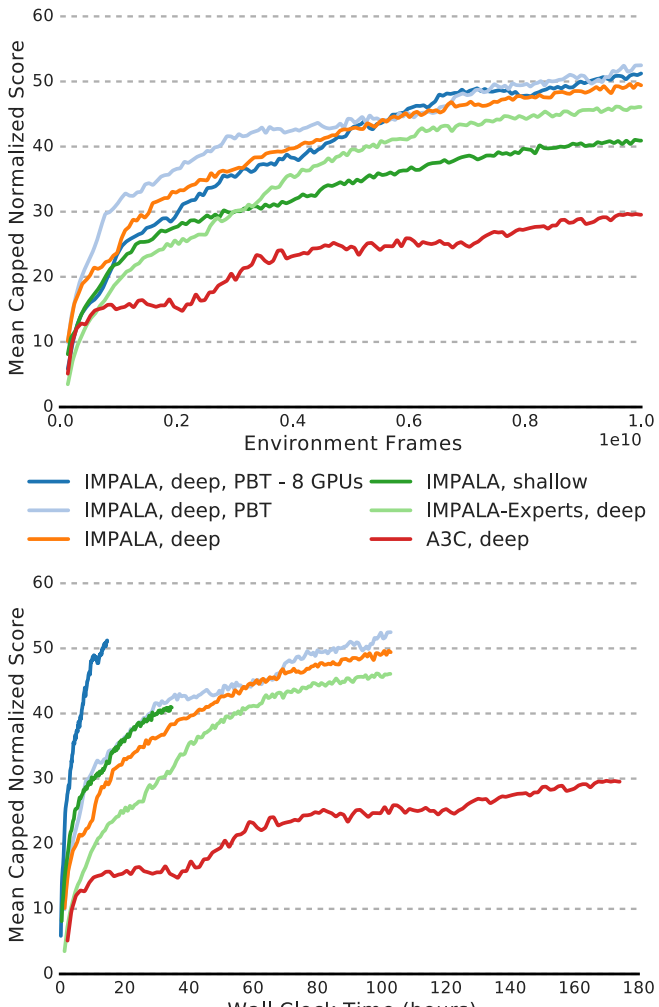


Figure 4 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

IMPALA – Learning Curves



Figures 5, 6 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

IMPALA – Atari Games

Human Normalised Return	Median	Mean
A3C, shallow, experts	54.9%	285.9%
A3C, deep, experts	117.9%	503.6%
Reactor, experts	187%	N/A
IMPALA, shallow, experts	93.2%	466.4%
IMPALA, deep, experts	191.8%	957.6%
IMPALA, deep, multi-task	59.7%	176.9%

Table 4 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

IMPALA – Ablations

	Task 1	Task 2	Task 3	Task 4	Task 5
Without Replay					
V-trace	46.8	32.9	**31.3**	**229.2**	**43.8**
1-Step	**51.8**	**35.9**	25.4	215.8	43.7
ε-correction	44.2	27.3	4.3	107.7	41.5
No-correction	40.3	29.1	5.0	94.9	16.1
With Replay					
V-trace	47.1	**35.8**	**34.5**	**250.8**	**46.9**
1-Step	**54.7**	34.4	26.4	204.8	41.6
ε-correction	30.4	30.2	3.9	101.5	37.6
No-correction	35.0	21.1	2.8	85.0	11.2

Tasks: `rooms_watermaze`, `rooms_keys_doors_puzzle`, `lasertag_three_opponents_small`, `explore_goal_locations_small`, `seekavoid_arena_01`

Table 2 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

IMPALA – Ablations

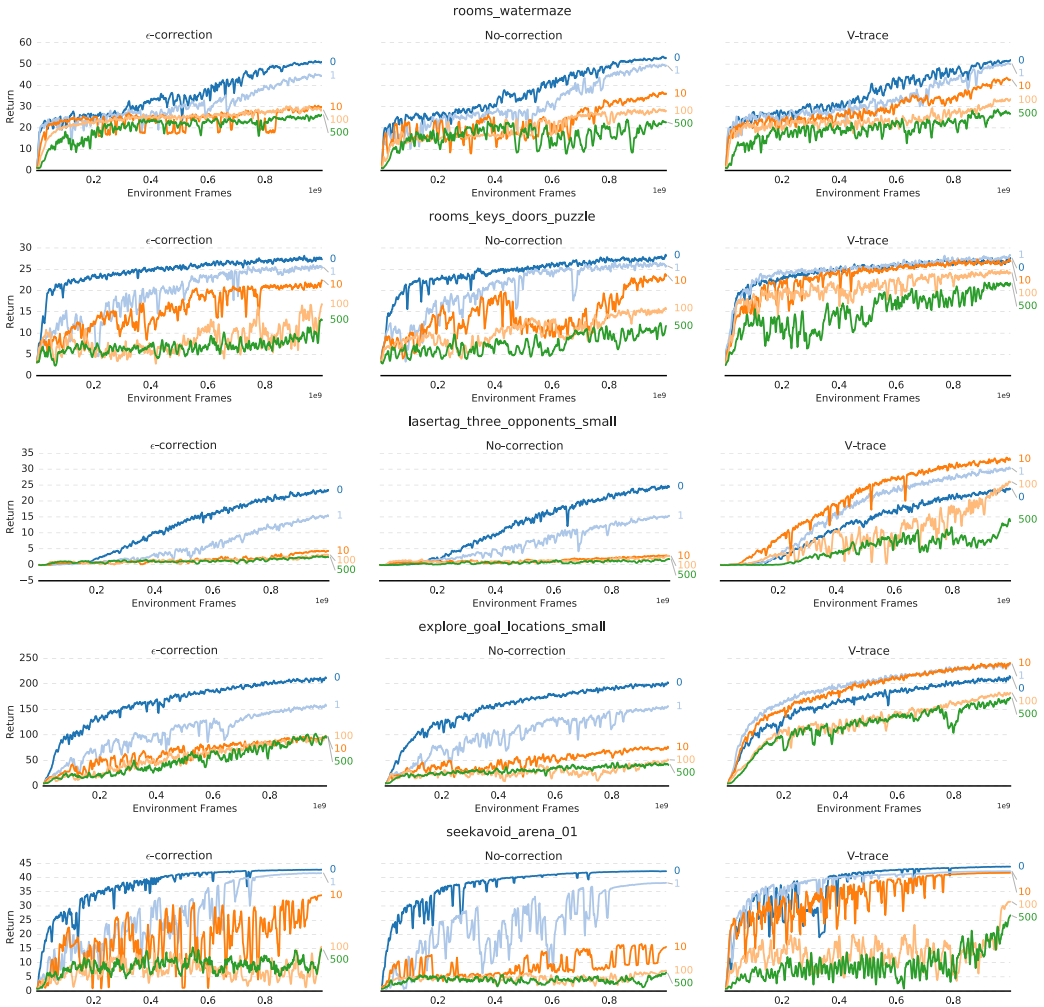


Figure E.1 of the paper "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures" by Lasse Espeholt et al.

PopArt Normalization

An improvement of IMPALA from Sep 2018, which performs normalization of task rewards instead of just reward clipping. PopArt stands for *Preserving Outputs Precisely, while Adaptively Rescaling Targets*.

Assume the value estimate $v(s; \boldsymbol{\theta}, \sigma, \mu)$ is computed using a normalized value predictor $n(s; \boldsymbol{\theta})$

$$v(s; \boldsymbol{\theta}, \sigma, \mu) \stackrel{\text{def}}{=} \sigma n(s; \boldsymbol{\theta}) + \mu$$

and further assume that $n(s; \boldsymbol{\theta})$ is an output of a linear function

$$n(s; \boldsymbol{\theta}) \stackrel{\text{def}}{=} \boldsymbol{\omega}^T f(s; \boldsymbol{\theta} - \{\boldsymbol{\omega}, b\}) + b.$$

We can update the σ and μ using exponentially moving average with decay rate β (in the paper, first moment μ and second moment v is tracked, and standard deviation is computed as $\sigma = \sqrt{v - \mu^2}$; decay rate $\beta = 3 \cdot 10^{-4}$ is employed).

PopArt Normalization

Utilizing the parameters μ and σ, we can normalize the observed (unnormalized) returns as $(G - \mu)/\sigma$ and use an actor-critic algorithm with advantage $(G - \mu)/\sigma - n(S; \boldsymbol{\theta})$.

However, in order to make sure the value function estimate does not change when the normalization parameters change, the parameters $\boldsymbol{\omega}, b$ computing the unnormalized value estimate are updated under any change $\mu \to \mu'$ and $\sigma \to \sigma'$ as:

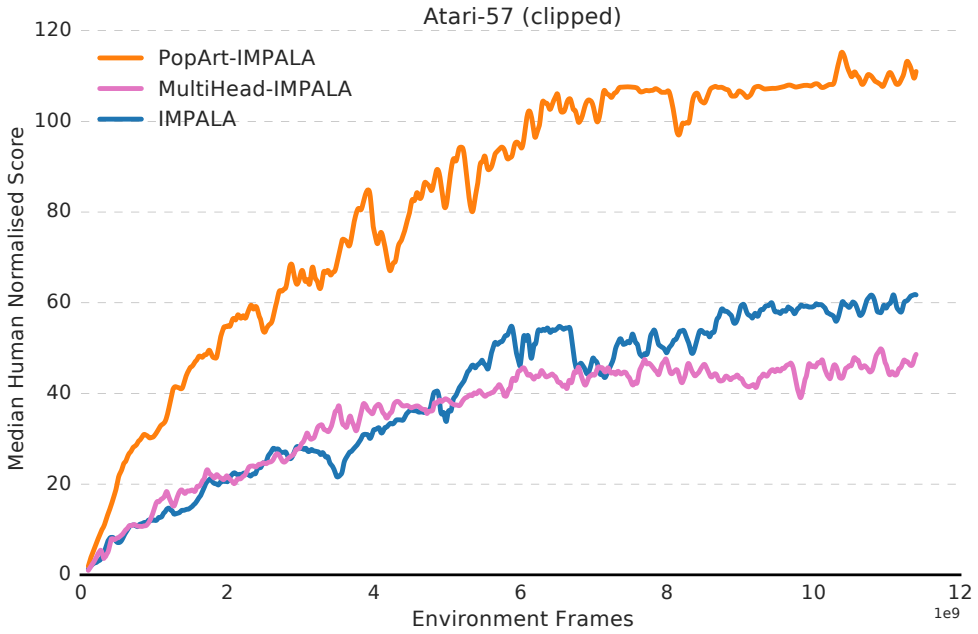
$$\boldsymbol{\omega}' \stackrel{\text{def}}{=} \frac{\sigma}{\sigma'}\boldsymbol{\omega}, \quad b' \stackrel{\text{def}}{=} \frac{\sigma b + \mu - \mu'}{\sigma'}.$$

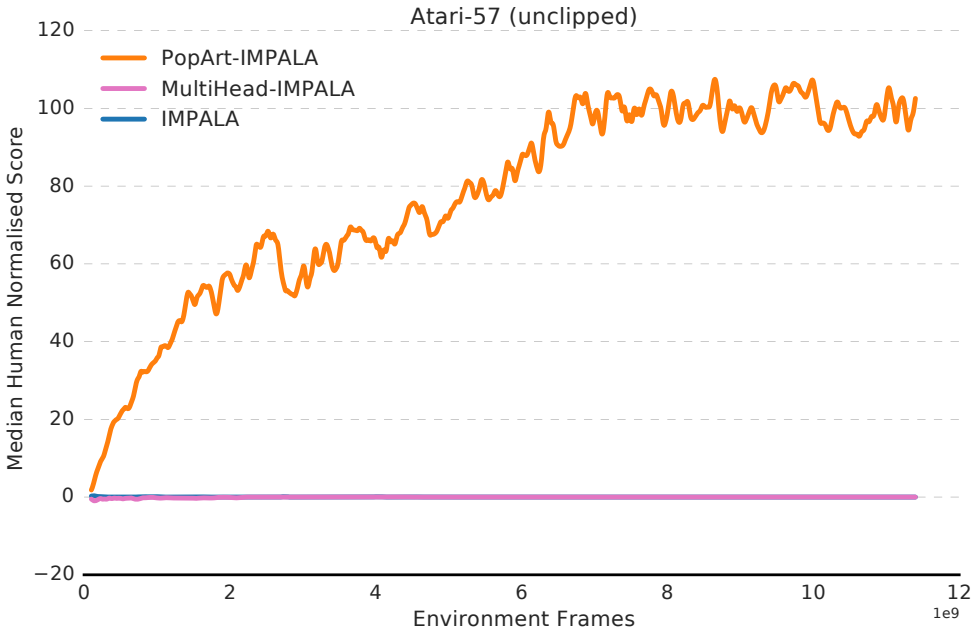
In multi-task settings, we train a task-agnostic policy and task-specific value functions (therefore, $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{n}(s; \boldsymbol{\theta})$ are vectors).

PopArt Results

	Atari-57		Atari-57 (unclipped)		DmLab-30	
Agent	Random	Human	Random	Human	Train	Test
IMPALA	59.7%	28.5%	0.3%	1.0%	60.6%	58.4%
PopArt-IMPALA	110.7%	101.5%	107.0%	93.7%	73.5%	72.8%

Table 1 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.





Figures 1, 2 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

PopArt Results

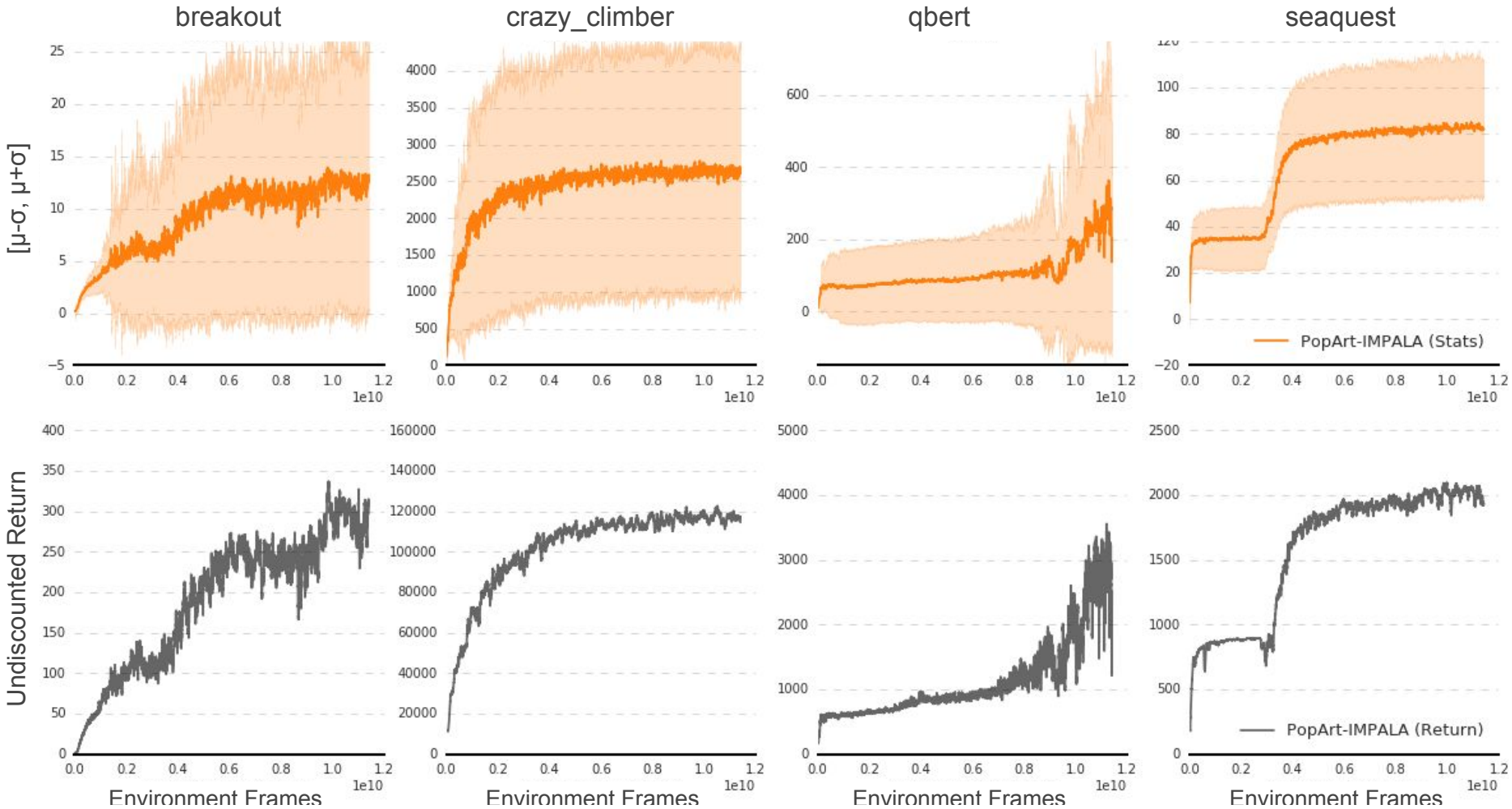
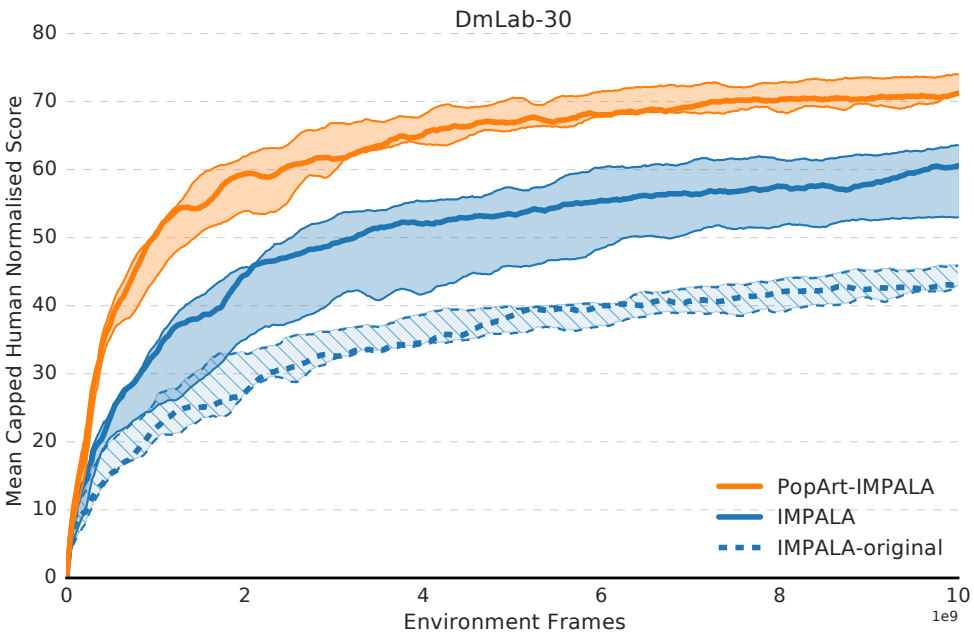
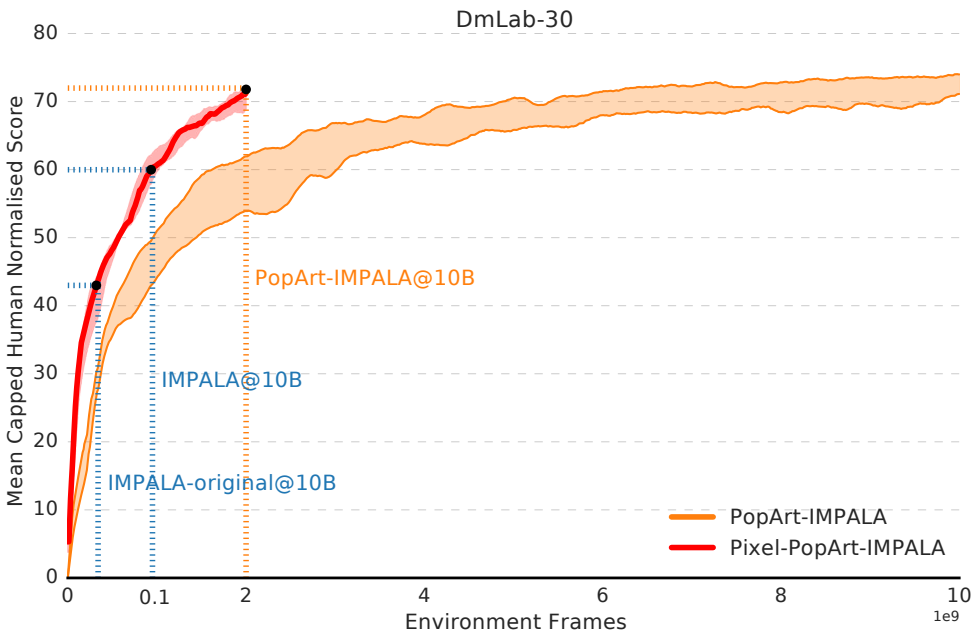


Figure 3 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

PopArt Results





Figures 4, 5 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

Partially Observable MDPs

Recall that a *Markov decision process* (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a *reward* $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a *discount factor*.

Partially observable Markov decision process extends the Markov decision process to a sextuple $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$, where in addition to an MDP

- $\mathcal{O}$ is a set of observations,
- $o(O_t | S_t, A_{t-1})$ is an observation model.

In robotics (out of the domain of this course), several approaches are used to handle POMDPs, to model uncertainty, imprecise mechanisms and inaccurate sensors.

Partially Observable MDPs

In Deep RL, partially observable MDPs are usually handled using recurrent networks. After suitable encoding of input observation O_t and previous action A_{t-1}, a RNN (usually LSTM) unit is used to model the current S_t (or its suitable latent representation), which is in turn utilized to produce A_t.

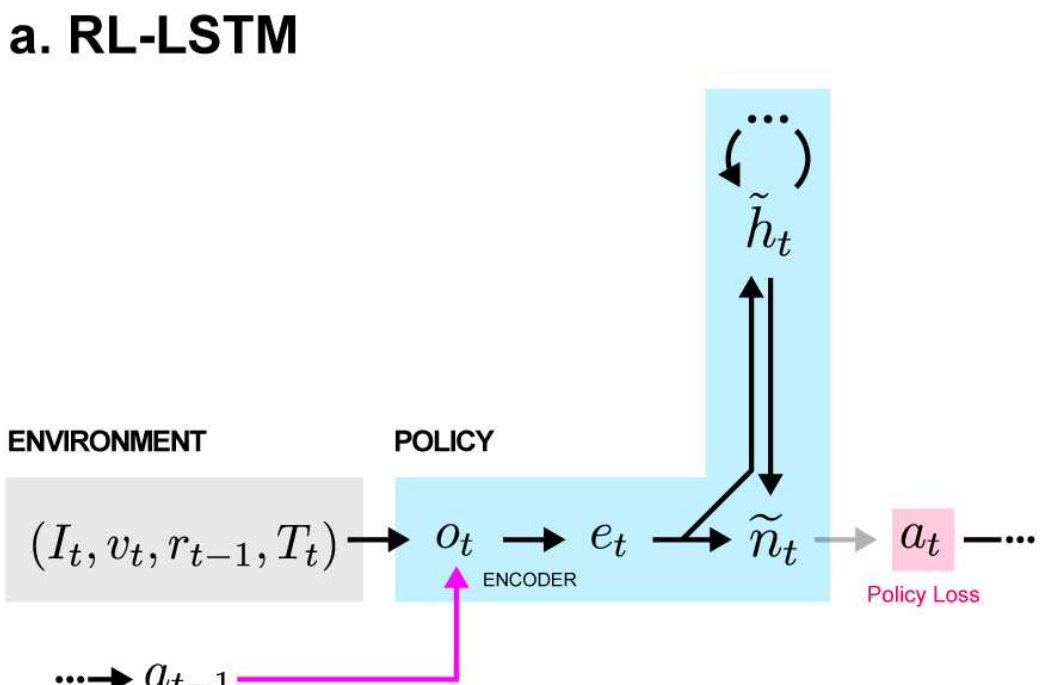


Figure 1a of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

However, keeping all information in the RNN state is substantially limiting. Therefore, *memory-augmented* networks can be used to store suitable information in external memory (in the lines of NTM, DNC or MANN models).

We now describe an approach used by Merlin architecture (*Unsupervised Predictive Memory in a Goal-Directed Agent* DeepMind Mar 2018 paper).

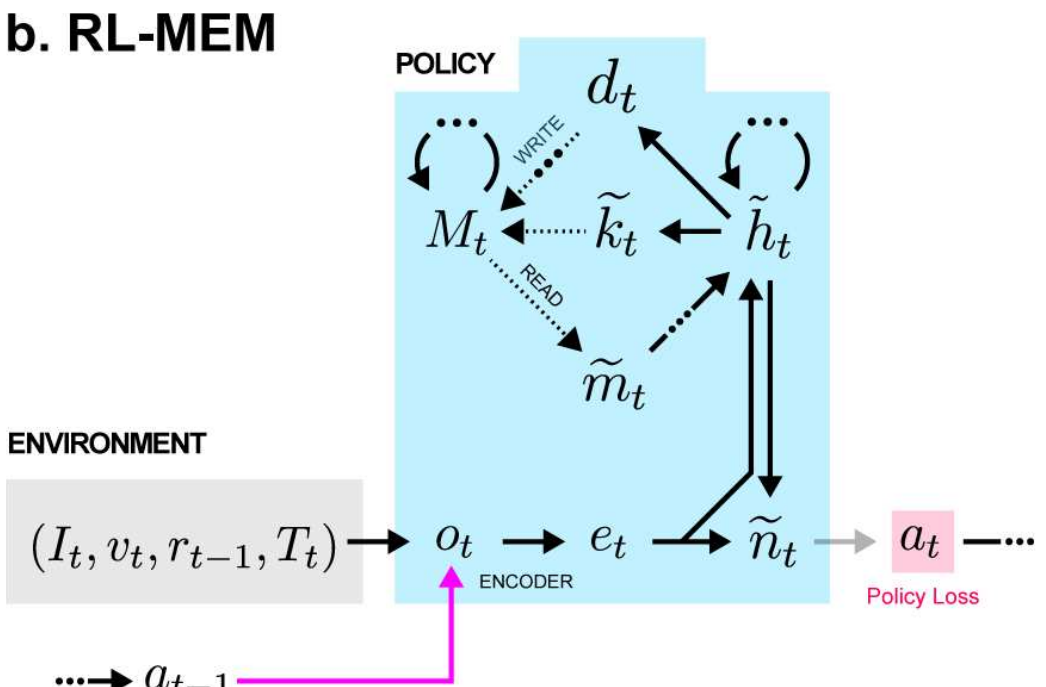


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

MERLIN – Memory Module

Let $\boldsymbol{M}$ be a memory matrix of size $N_{mem} \times 2|z|$.

Assume we have already encoded observations as $\boldsymbol{e}_t$ and previous action a_{t-1}. We concatenate them with K previously read vectors and process by a deep LSTM (two layers are used in the paper) to compute $\boldsymbol{h}_t$.

Then, we apply a linear layer to $\boldsymbol{h}_t$, computing K key vectors $\boldsymbol{k}_1, \ldots \boldsymbol{k}_K$ of length $2|z|$ and K positive scalars $\beta_1, \ldots, \beta_K$.

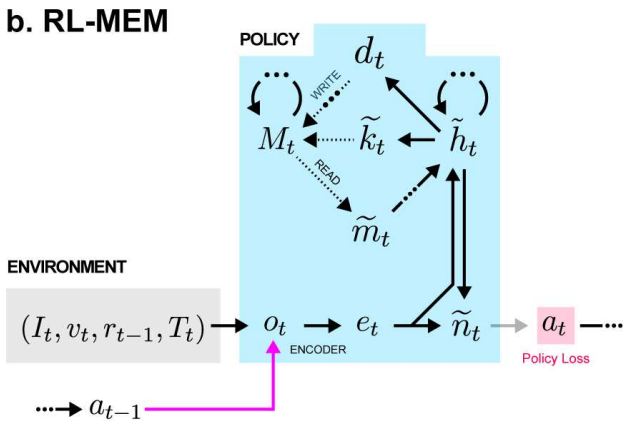


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

Reading: For each i, we compute cosine similarity of $\boldsymbol{k}_i$ and all memory rows M_j, multiply the similarities by β_i and pass them through a $\mathrm{softmax}$ to obtain weights $\boldsymbol{\omega}_i$. The read vector is then computed as $\boldsymbol{M}\boldsymbol{w}_i$.

Writing: We find one-hot write index $\boldsymbol{v}_{wr}$ to be the least used memory row (we keep usage indicators and add read weights to them). We then compute $\boldsymbol{v}_{ret} \leftarrow \gamma \boldsymbol{v}_{ret} + (1-\gamma)\boldsymbol{v}_{wr}$, and update the memory matrix using $\boldsymbol{M} \leftarrow \boldsymbol{M} + \boldsymbol{v}_{wr}[\boldsymbol{e}_t, 0] + \boldsymbol{v}_{ret}[0, \boldsymbol{e}_t]$.

MERLIN — Prior and Posterior

However, updating the encoder and memory content purely using RL is inefficient. Therfore, MERLIN includes a *memory-based predictor (MBP)* in addition to policy. The goal of MBP is to compress observations into low-dimensional state representations z and storing them in memory.

According to the paper, the idea of unsupervised and predictive modeling has been entertained for decades, and recent discussions have proposed such modeling to be connected to hippocampal memory.

We want the state variables not only to faithfully represent the data, but also emphasise rewarding elements of the environment above irrelevant ones. To accomplish this, the authors follow the hippocampal representation theory of Gluck and Myers, who proposed that hippocampal representations pass through a compressive bottleneck and then reconstruct input stimuli together with task reward.

In MERLIN, a *prior* distribution over z_t predicts next state variable conditioned on history of state variables and actions $p(z_t|z_{t-1}, a_{t-1}, \ldots, z_1, a_1)$, and *posterior* corrects the prior using the new observation o_t, forming a better estimate $q(z_t|o_t, z_{t-1}, a_{t-1}, \ldots, z_1, a_1)$.

MERLIN — Prior and Posterior

To achieve the mentioned goals, we add two terms to the loss.

- We try reconstructing input stimuli, action, reward and return using a sample from the state variable posterior, and add the difference of the reconstruction and ground truth to the loss.
- We also add KL divergence of the prior and posterior to the loss, to ensure consistency between the prior and posterior.

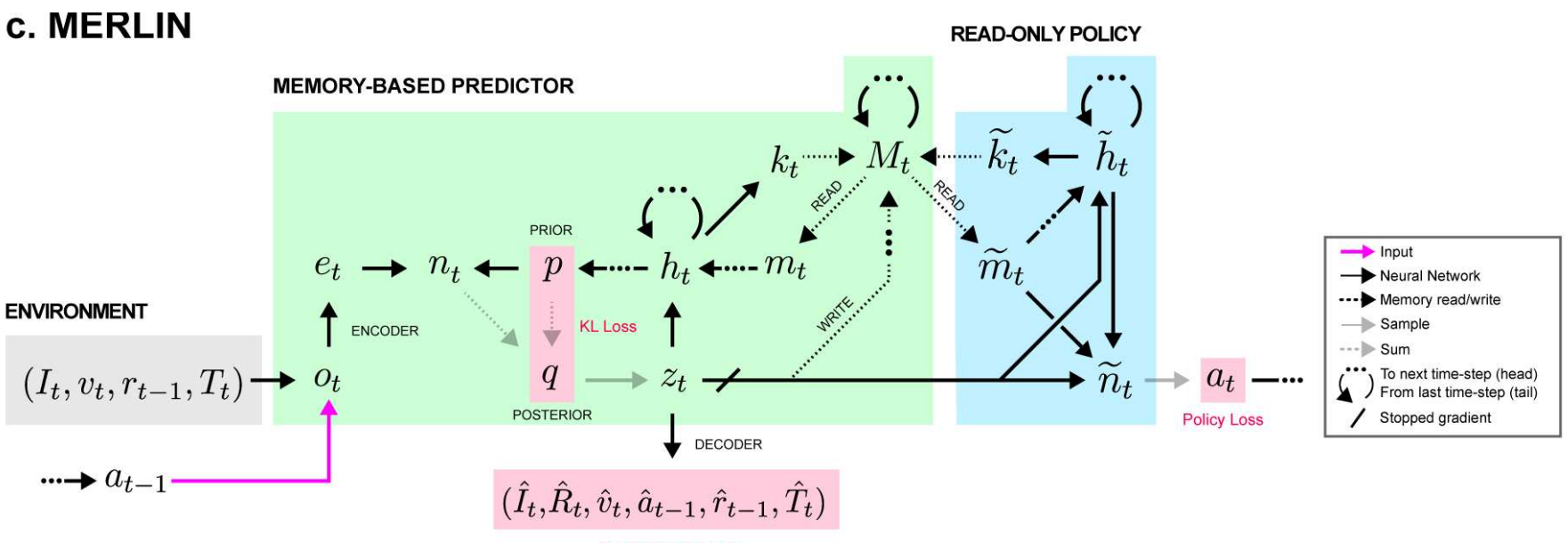


Figure 1c of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

MERLIN — Algorithm

Algorithm 1 MERLIN Worker Pseudocode

// Assume global shared parameter vectors θ for the policy network and χ for the memory-based predictor; global shared counter $T := 0$
// Assume thread-specific parameter vectors θ', χ'
// Assume discount factor $\gamma \in (0, 1]$ and bootstrapping parameter $\lambda \in [0, 1]$
Initialize thread step counter $t := 1$
repeat
  Synchronize thread-specific parameters $\theta' := \theta; \chi' := \chi$
  Zero model's memory & recurrent state if new episode begins
  $t_{\text{start}} := t$
  repeat
    Prior $\mathcal{N}(\mu_t^{\text{p}}, \log \Sigma_t^{\text{p}}) = p(h_{t-1}, m_{t-1})$
    $e_t = \text{enc}(o_t)$
    Posterior $\mathcal{N}(\mu_t^{\text{q}}, \log \Sigma_t^{\text{q}}) = q(e_t, h_{t-1}, m_{t-1}, \mu_t^{\text{p}}, \log \Sigma_t^{\text{p}})$
    Sample $z_t \sim \mathcal{N}(\mu_t^{\text{q}}, \log \Sigma_t^{\text{q}})$
    Policy network update $\tilde{h}_t = \text{rec}(\tilde{h}_{t-1}, \tilde{m}_t, \text{StopGradient}(z_t))$
    Policy distribution $\pi_t = \pi(\tilde{h}_t, \text{StopGradient}(z_t))$
    Sample $a_t \sim \pi_t$
    $h_t = \text{rec}(h_{t-1}, m_t, z_t)$
    Update memory with z_t by Methods Eq. 2
    $R_t, o_t^r = \text{dec}(z_t, \pi_t, a_t)$
    Apply a_t to environment and receive reward r_t and observation o_{t+1}
    $t := t + 1; T := T + 1$
  until environment termination or $t - t_{\text{start}} == \tau_{\text{window}}$
  If not terminated, run additional step to compute $V_\nu^\pi(z_{t+1}, \log \pi_{t+1})$
  and set $R_{t+1} := V^\pi(z_{t+1}, \log \pi_{t+1})$ // (but don't increment counters)
  Reset performance accumulators $\mathcal{A} := 0; \mathcal{L} := 0; \mathcal{H} := 0$
  for k from t down to t_{start} **do**
    $$\gamma_t := \begin{cases} 0, & \text{if } k \text{ is environment termination} \\ \gamma, & \text{otherwise} \end{cases}$$
    $R_k := r_k + \gamma_t R_{k+1}$
    $\delta_k := r_k + \gamma_t V^\pi(z_{k+1}, \log \pi_{k+1}) - V^\pi(z_k, \log \pi_k)$
    $A_k := \delta_k + (\gamma\lambda) A_{k+1}$
    $\mathcal{L} := \mathcal{L} + \mathcal{L}_k$ (Eq. 7)
    $\mathcal{A} := \mathcal{A} + A_k \log \pi_k[a_k]$
    $\mathcal{H} := \mathcal{H} - \alpha_{\text{entropy}} \sum_i \pi_k[i] \log \pi_k[i]$ (Entropy loss)
  end for
  $d\chi' := \nabla_{\chi'} \mathcal{L}$
  $d\theta' := \nabla_{\theta'}(\mathcal{A} + \mathcal{H})$
  Asynchronously update via gradient ascent θ using $d\theta'$ and χ using $d\chi'$
until $T > T_{\text{max}}$

Algorithm 1 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

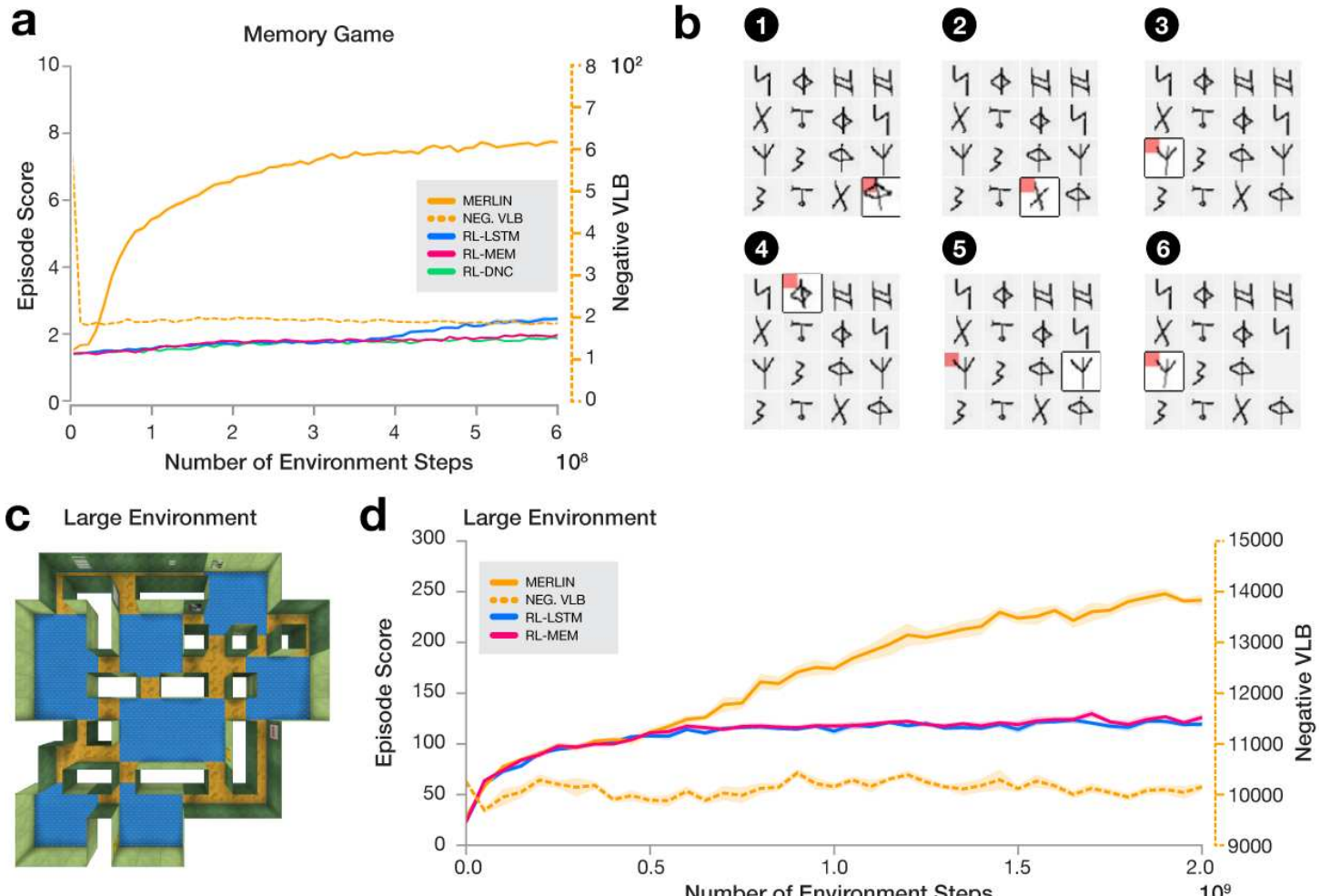


Figure 2 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

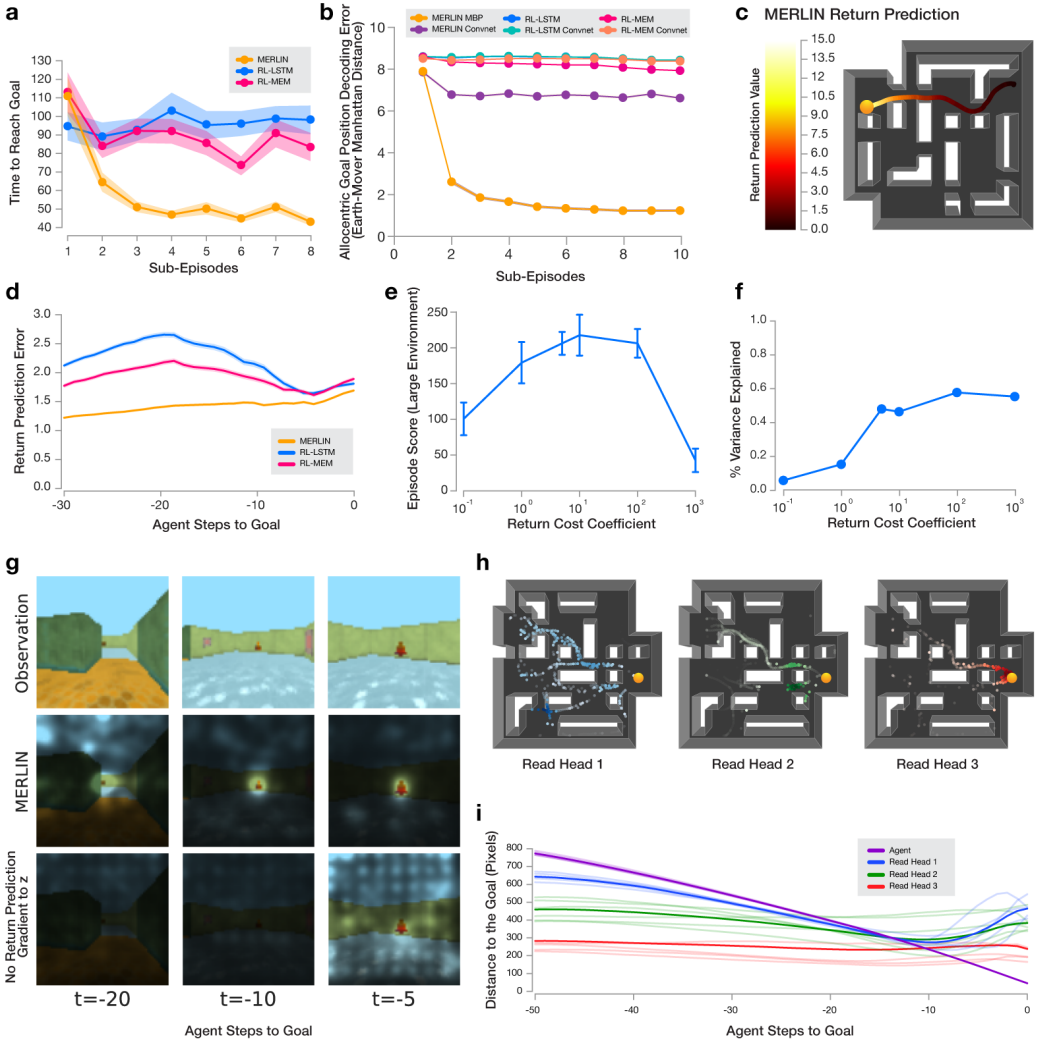
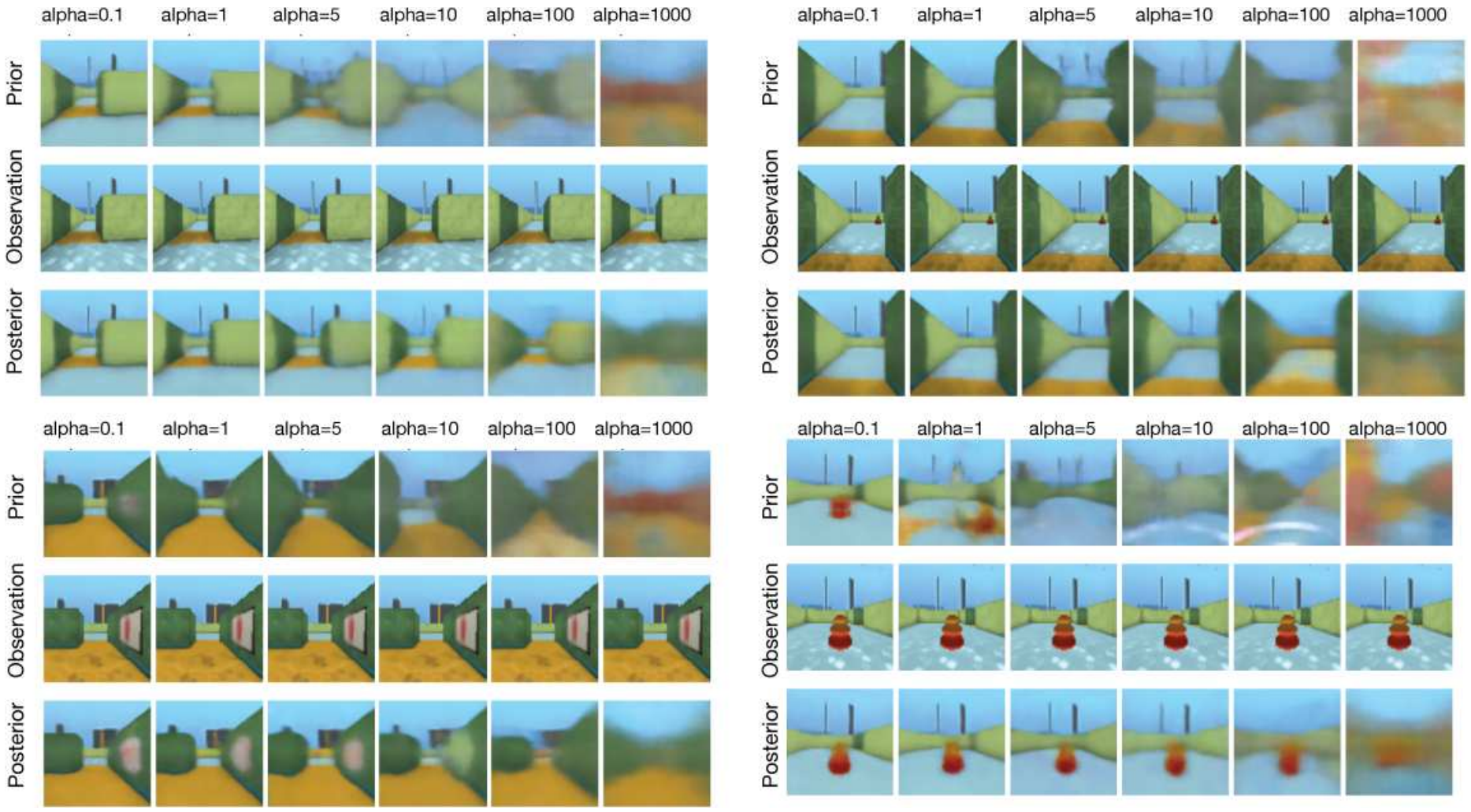


Figure 3 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

MERLIN



Extended Figure 3 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

For the Win agent for Capture The Flag

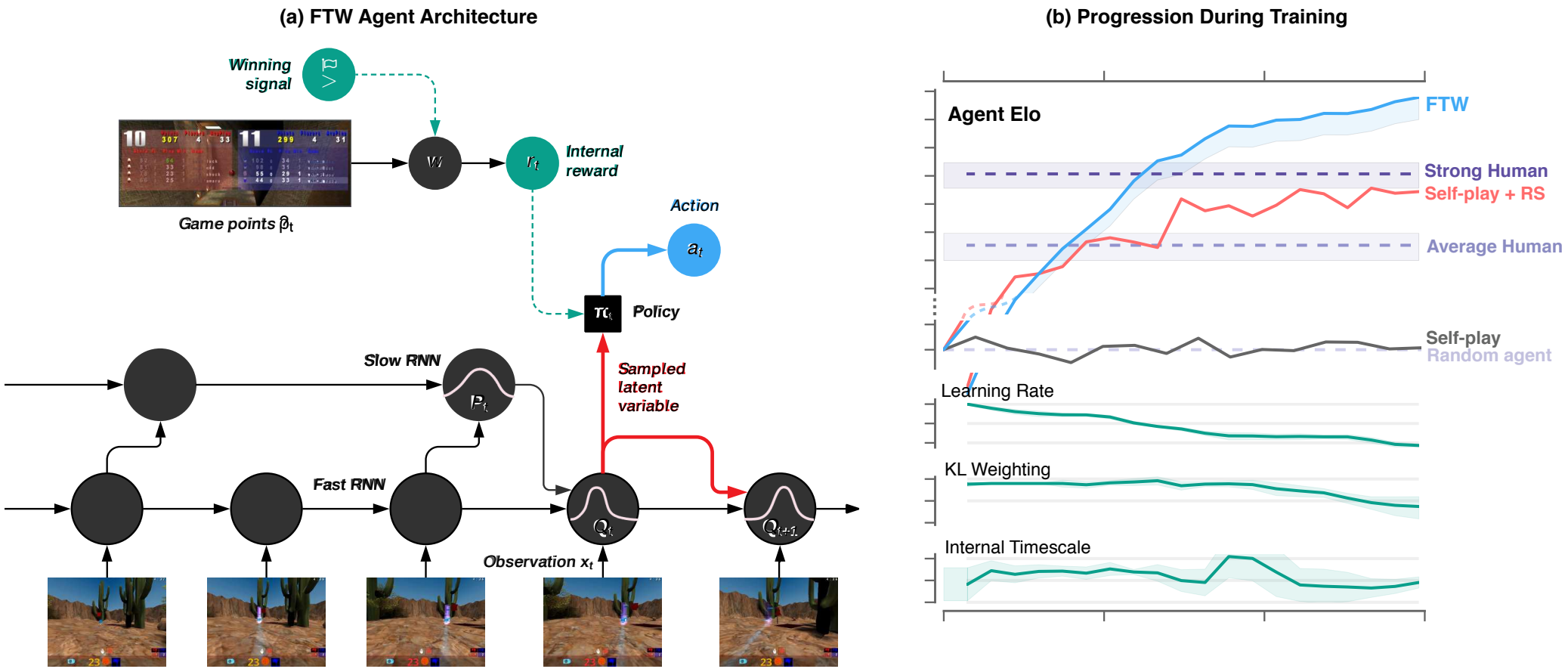


Figure 2 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

- Extension of the MERLIN architecture.
- Hierarchical RNN with two timescales.
- Population based training controlling KL divergence penalty weights, slow ticking RNN speed and gradient flow factor from fast to slow RNN.

For the Win agent for Capture The Flag

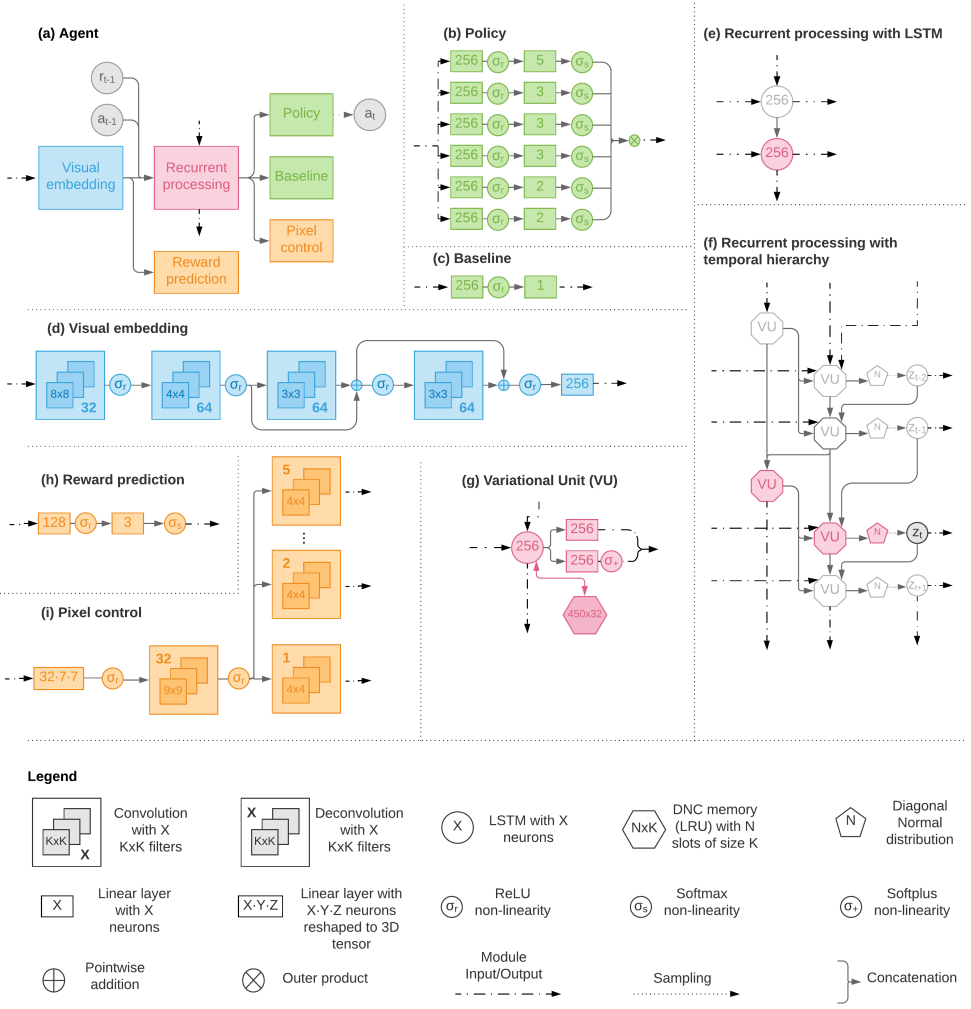


Figure S10 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

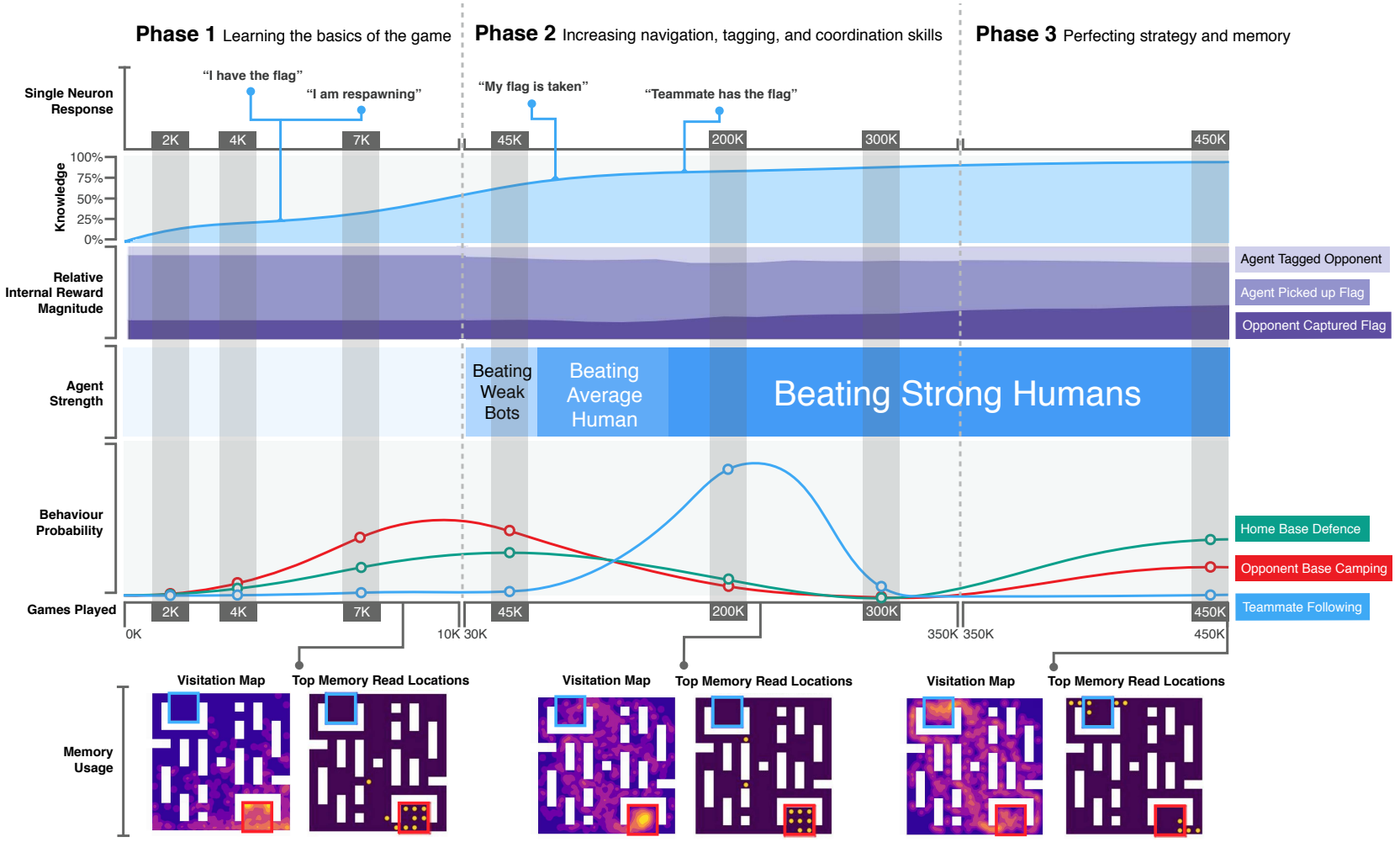


Figure 4 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.