ST and Gumbel-Softmax, DreamerV2, MERLIN, FTW

Milan Straka

December 20, 2021

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

Discrete Latent Variables

Consider that we would like to have discrete neurons on the hidden layer of a neural network.

Note that on the output layer, we relaxed discrete prediction (i.e., an $\arg\max$) with a continuous relaxation – $\operatorname{softmax}$. This way, we can compute the derivatives, and also predict the most probable class.

However, on a hidden layer, we also need to *sample* from the predicted categorical distribution, and then backpropagate the gradients.

Stochastic Gradient Estimators

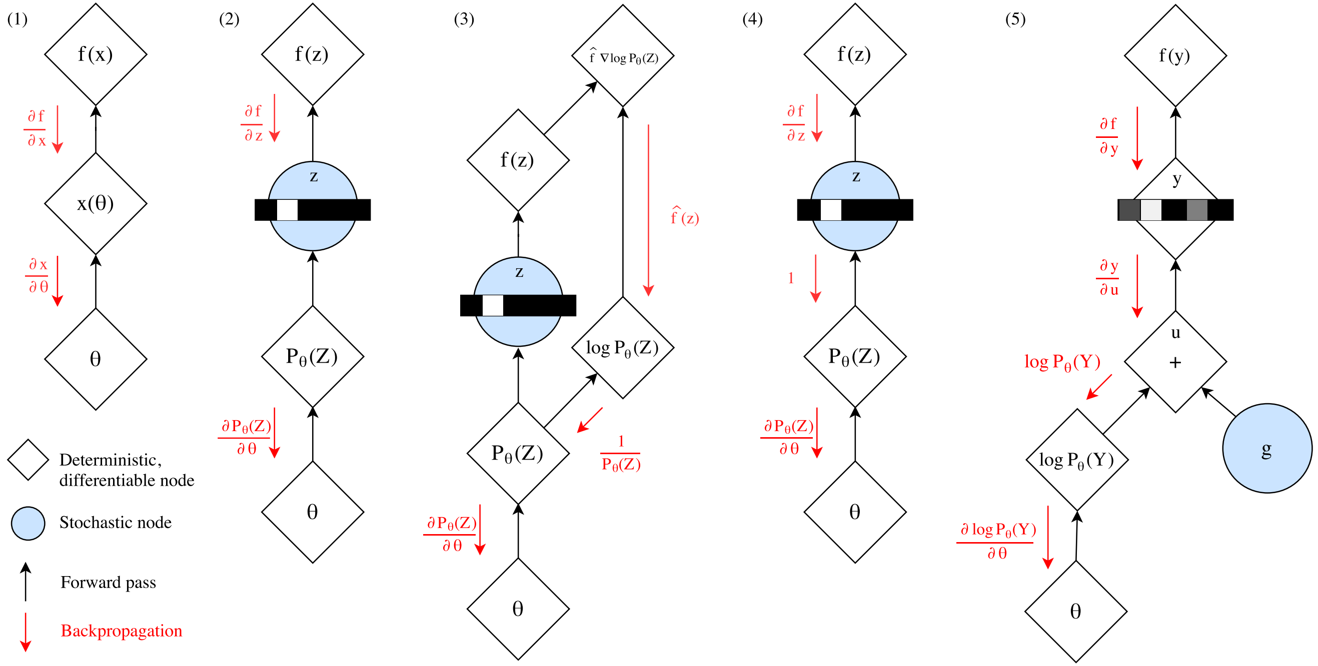


Figure 2: Gradient estimation in stochastic computation graphs. (1) $\nabla_\theta f(x)$ can be computed via backpropagation if $x(\theta)$ is deterministic and differentiable. (2) The presence of stochastic node z precludes backpropagation as the sampler function does not have a well-defined gradient. (3) The score function estimator and its variants (NVIL, DARN, MuProp, VIMCO) obtain an unbiased estimate of $\nabla_\theta f(x)$ by backpropagating along a surrogate loss $\hat{f} \log p_\theta(z)$, where $\hat{f} = f(x) - b$ and b is a baseline for variance reduction. (4) The Straight-Through estimator, developed primarily for Bernoulli variables, approximates $\nabla_\theta z \approx 1$. (5) Gumbel-Softmax is a path derivative estimator for a continuous distribution y that approximates z. Reparameterization allows gradients to flow from $f(y)$ to θ. y can be annealed to one-hot categorical variables over the course of training.

Figure 2 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Consider a model with a discrete categorical latent variable $\boldsymbol{z}$ sampled from $p(\boldsymbol{z};\boldsymbol{\theta})$, with a loss $L(\boldsymbol{z};\boldsymbol{\omega})$. Several gradient estimators have been proposed:

- A REINFORCE-like gradient estimation.

  Using the identity $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{z};\boldsymbol{\theta}) = p(\boldsymbol{z};\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})$, we obtain that

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{z}}\big[L(\boldsymbol{z};\boldsymbol{\omega})\big] = \mathbb{E}_{\boldsymbol{z}}\big[L(\boldsymbol{z};\boldsymbol{\omega})\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})\big].$$

  Analogously as before, we can also include the baseline for variance reduction, resulting in

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{z}}\big[L(\boldsymbol{z};\boldsymbol{\omega})\big] = \mathbb{E}_{\boldsymbol{z}}\big[(L(\boldsymbol{z};\boldsymbol{\omega}) - b)\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})\big].$$

- A **straight-through (ST)** estimator.

  The straight-through estimator has been proposed by Y. Bengio in 2013. It is a biased estimator, which assumes that $\nabla_{\boldsymbol{\theta}}\boldsymbol{z} \approx \nabla_{\boldsymbol{\theta}} p(\boldsymbol{z};\boldsymbol{\theta})$, which implies $\nabla_{p(\boldsymbol{z};\boldsymbol{\theta})}\boldsymbol{z} \approx 1$. Even if the bias can be considerable, it seems to work quite well in practice.

The **Gumbel-softmax** distribution was proposed independently in two papers in Nov 2016 (under the name of **Concrete** distribution in the other paper).

It is a continuous distribution over the simplex (over categorical distributions) that can approximate *sampling* from a categorical distribution.

Let z be a categorical variable with class probabilities $\boldsymbol{p} = (p_1, p_2, \ldots, p_K)$.

The Gumbel-Max trick (based on a 1954 theorem from E. J. Gumbel) states that we can draw samples $z \sim \boldsymbol{p}$ using

$$z = \operatorname{one-hot}\Big(\arg\max_i \big(g_i + \log p_i\big)\Big),$$

where g_i are independent samples drawn from the $\operatorname{Gumbel}(0, 1)$ distribution.

To sample g from the distribution $\operatorname{Gumbel}(0, 1)$, we can sample $u \sim U(0, 1)$ and then compute $g = -\log(-\log u)$.

Gumbel-Max Trick Proof

To prove the Gumbel-Max trick, we first reformulate it slightly.

Let l_i be logits of a categorical distribution (so that the class probabilities $\pi_i \propto e^{l_i}$), and let $g_i \sim \operatorname{Gumbel}(0, 1)$. Then

$$\pi_k = P\big(k = \arg\max_i (g_i + l_i)\big).$$

We first observe that the theorem is invariant to a scalar shift of logits, so we can without loss of generality assume that $\sum_i e^{l_i} = 1$ and $\pi_i = e^{l_i}$.

For convenience, denote $u_i \stackrel{\text{def}}{=} g_i + l_i$.

Recall that the PDF and CDF of a $\operatorname{Gumbel}(0, 1)$ distribution are:

$$\mathrm{PDF}(g_i) = e^{-g_i - e^{-g_i}},$$

$$\mathrm{CDF}(g_i) = e^{-e^{-g_i}}.$$

Gumbel-Max Trick Proof

To finish the proof, we compute

$$
\begin{aligned}
P\big(k = \arg\max_i (g_i + l_i)\big) &= P(u_k \geq u_i, \forall_{i \neq k}) \\
&= \int P(u_k) \prod\nolimits_{i \neq k} P(u_k \geq u_i | u_k) \, \mathrm{d}u_k \\
&= \int P(g_k | g_k = u_k - l_k) \prod\nolimits_{i \neq k} P(g_i \leq u_k - l_i | u_k) \, \mathrm{d}u_k \\
&= \int e^{l_k - u_k - e^{l_k - u_k}} \prod\nolimits_{i \neq k} e^{-e^{l_i - u_k}} \, \mathrm{d}u_k \\
&= \int \pi_k e^{-u_k - \pi_k e^{-u_k}} \prod\nolimits_{i \neq k} e^{-\pi_i e^{-u_k}} \, \mathrm{d}u_k \\
&= \pi_k \int e^{-u_k - e^{-u_k} \sum_i \pi_i} \, \mathrm{d}u_k \\
&= \pi_k \int e^{-g_k - e^{-g_k}} \, \mathrm{d}g_k = \pi_k \cdot 1 = \pi_k.
\end{aligned}
$$

To obtain a continuous distribution, we relax the $\arg\max$ into a $\operatorname{softmax}$ with temperature T as

$$z_i = \frac{e^{(g_i + \log p_i)/T}}{\sum_j e^{(g_j + \log p_j)/T}}.$$

As the temperature T goes to zero, the generated samples become one-hot and therefore the Gumbel-softmax distribution converges to the categorical distribution $p(z)$.

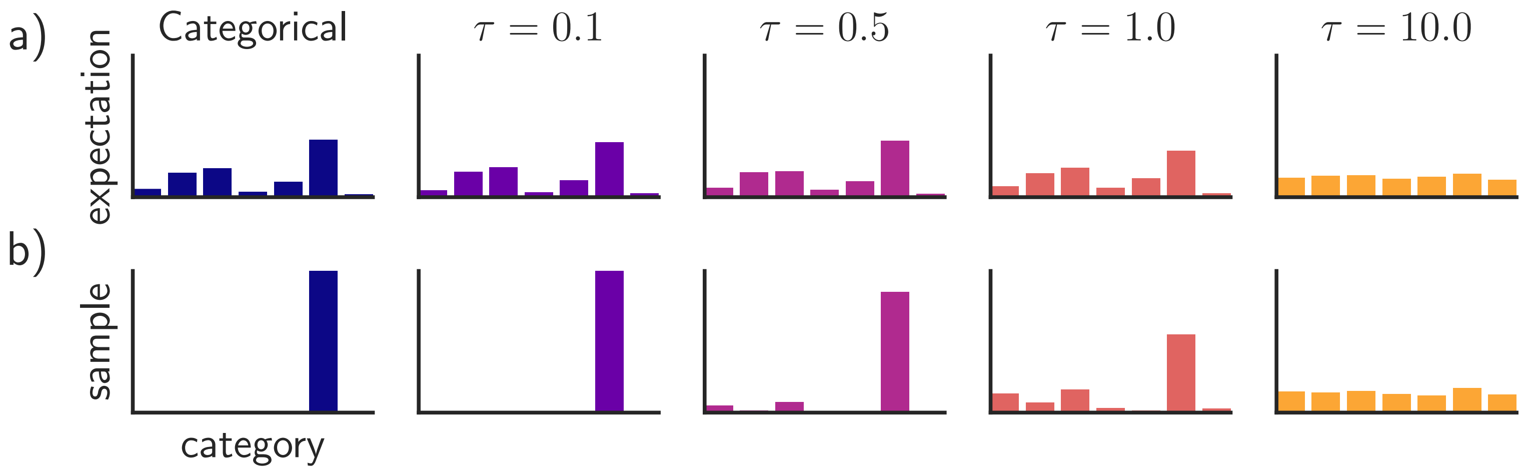


Figure 1 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Gumbel-Softmax Estimator

The Gumbel-softmax distribution can be used to reparametrize the sampling of the discrete variable using a fully differentiable estimator.

However, the resulting sample is not discrete, it only converges to a discrete sample as the temperature T goes to zero.

If it is a problem, we can combine the Gumbel-softmax with a straight-through estimator, obtaining ST Gumbel-softmax, where we:

- discretize $\boldsymbol{y}$ as $\boldsymbol{z} = \arg\max \boldsymbol{y}$,
- assume $\nabla_{\boldsymbol{\theta}} \boldsymbol{z} \approx \nabla_{\boldsymbol{\theta}} \boldsymbol{y}$, or in other words, $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \approx 1$.

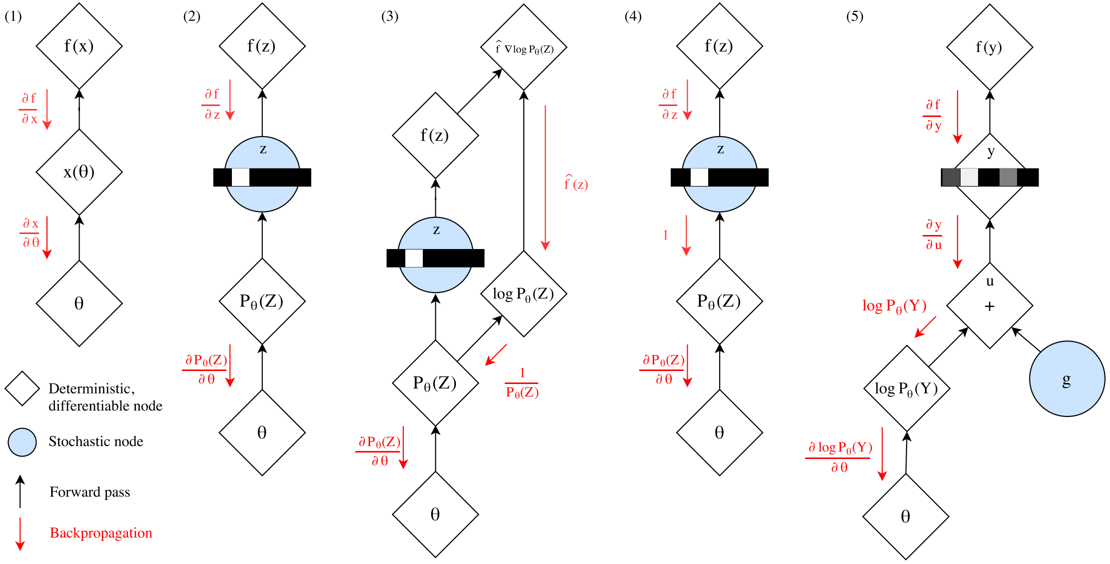


Figure 2: Gradient estimation in stochastic computation graphs. (1) $\nabla_\theta f(x)$ can be computed via backpropagation if $x(\theta)$ is deterministic and differentiable. (2) The presence of stochastic node z precludes backpropagation as the sampler function does not have a well-defined gradient. (3) The score function estimator and its variants (NVIL, DARN, MuProp, VIMCO) obtain an unbiased estimate of $\nabla_\theta f(x)$ by backpropagating along a surrogate loss $\hat{f} \log p_\theta(z)$, where $\hat{f} = f(x) - b$ and b is a baseline for variance reduction. (4) The Straight-Through estimator, developed primarily for Bernoulli variables, approximates $\nabla_\theta z \approx 1$. (5) Gumbel-Softmax is a path derivative estimator for a continuous distribution y that approximates z. Reparameterization allows gradients to flow from $f(y)$ to θ. y can be annealed to one-hot categorical variables over the course of training.

Figure 2 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Gumbel-Softmax Estimator Results

Table 1: The Gumbel-Softmax estimator outperforms other estimators on Bernoulli and Categorical latent variables. For the structured output prediction (SBN) task, numbers correspond to negative log-likelihoods (nats) of input images (lower is better). For the VAE task, numbers correspond to negative variational lower bounds (nats) on the log-likelihood (lower is better).

	SF	DARN	MuProp	ST	Annealed ST	Gumbel-S.	ST Gumbel-S.
SBN (Bern.)	72.0	59.7	58.9	58.9	58.7	**58.5**	59.3
SBN (Cat.)	73.1	67.9	63.0	61.8	61.1	**59.0**	59.7
VAE (Bern.)	112.2	110.9	109.7	116.0	111.5	**105.0**	111.5
VAE (Cat.)	110.6	128.8	107.0	110.9	107.8	**101.5**	107.8

Table 1 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

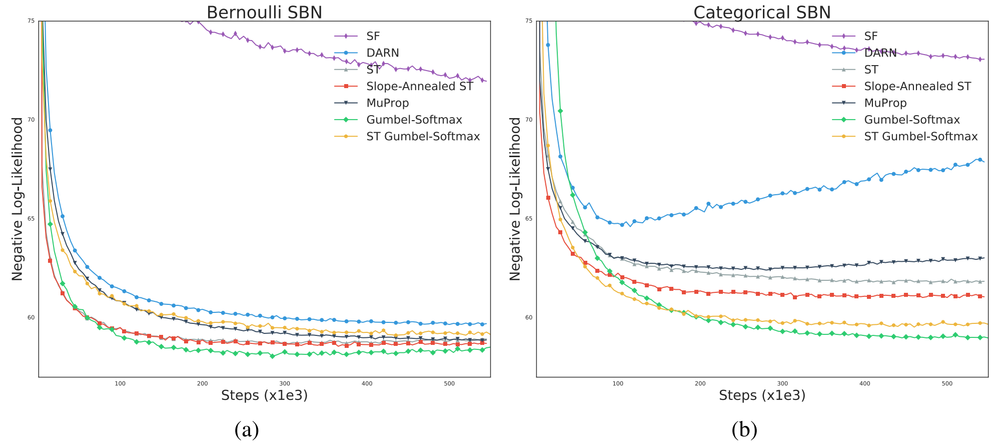


Figure 3: Test loss (negative log-likelihood) on the structured output prediction task with binarized MNIST using a stochastic binary network with (a) Bernoulli latent variables (392-200-200-392) and (b) categorical latent variables (392-(20×10)-(20×10)-392).

Figure 3 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

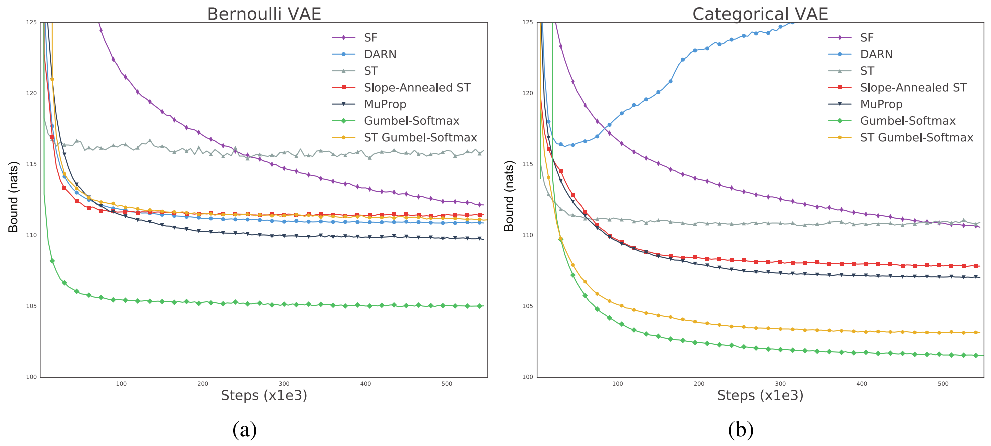


Figure 4: Test loss (negative variational lower bound) on binarized MNIST VAE with (a) Bernoulli latent variables $(784 - 200 - 784)$ and (b) categorical latent variables $(784 - (20 \times 10) - 200)$.

Figure 4 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

The discrete latent variables can be used among others to:

- allow the SAC algorithm to be used on **discrete** actions, using either Gumbel-softmax relaxation (if the critic takes the actions as binary indicators, it is possible to pass not just one-hot encoding, but the result of Gumbel-softmax directly), or a straight-through estimator;
- model images using discrete latent variables
  - VQ-VAE, VQ-VAE-2 use "codebook loss" with a straight-through estimator

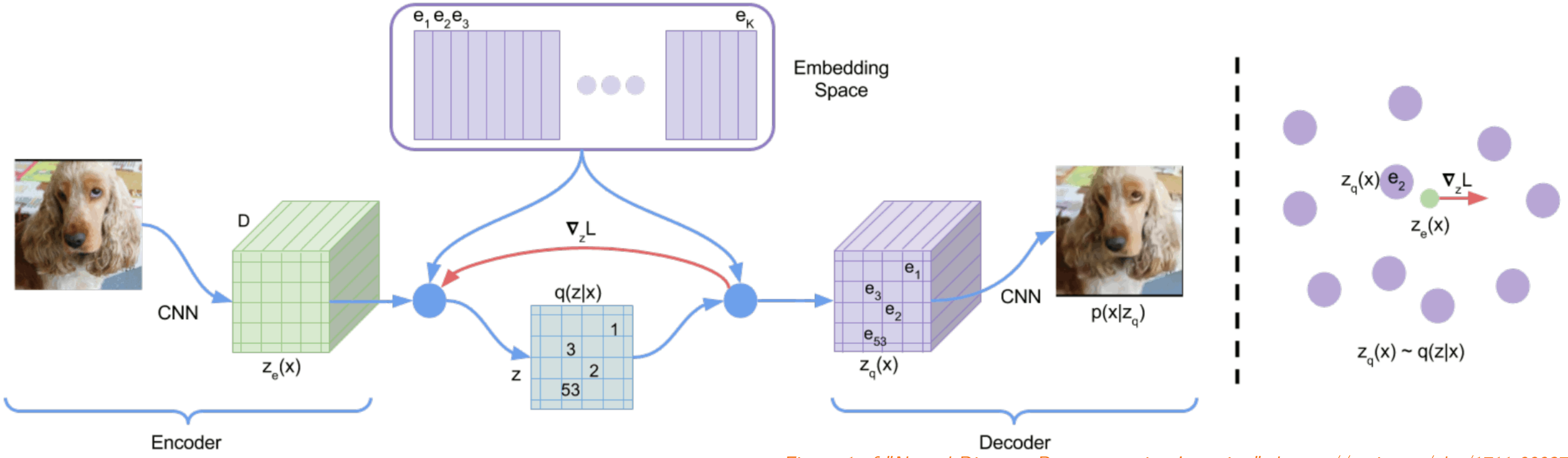


Figure 1 of "Neural Discrete Representation Learning", https://arxiv.org/abs/1711.00937

- VQ-GAN combines the VQ-VAE and Transformers, where the latter is used to generate a sequence of the *discrete* latents.

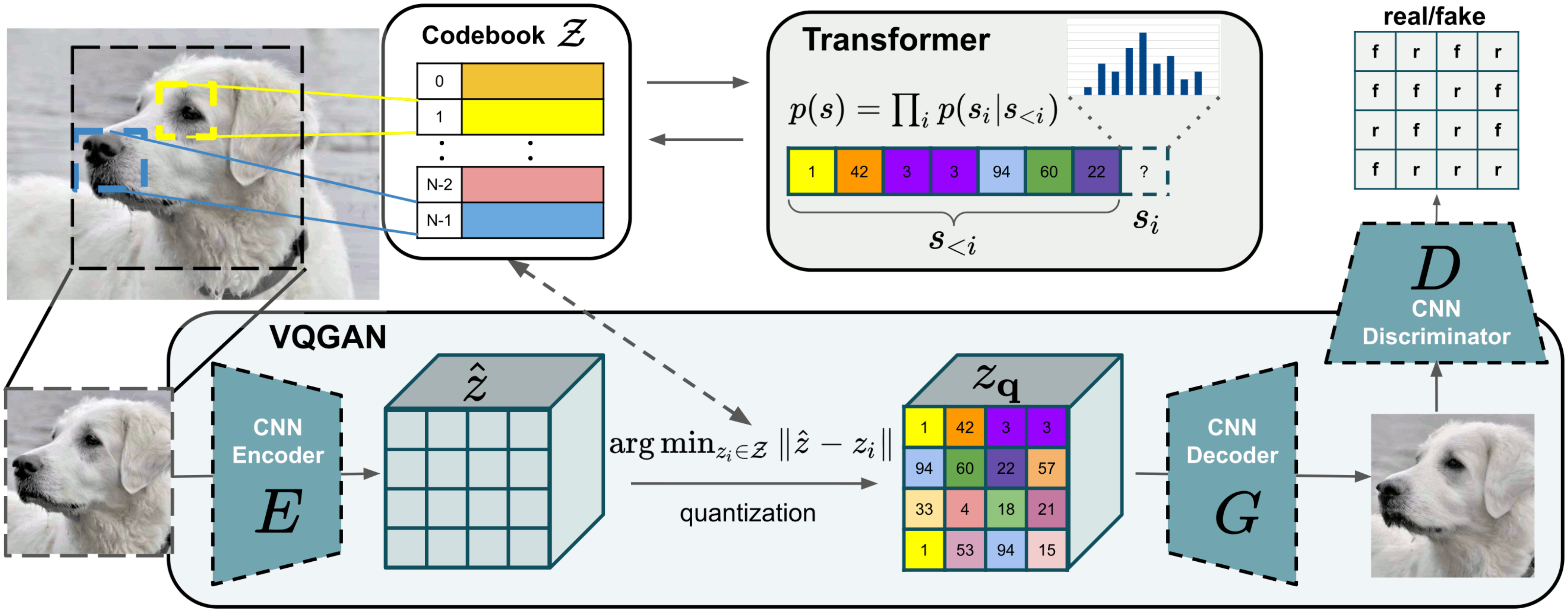


Figure 2 of "Taming Transformers for High-Resolution Image Synthesis", https://arxiv.org/abs/2012.09841

Applications of Discrete Latent Variables – VQ-GAN

Applications of Discrete Latent Variables – DALL-E

- In DALL-E, Transformer is used to model a sequence of words followed by a sequence of the discrete image latent variables.

  The Gumbel-softmax relaxation is used to train the discrete latent states, with temperature annealed with a cosine decay from 1 to 1/16 over the first 150k (out of 3M) updates.

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 2 of "Zero-Shot Text-to-Image Generation", https://arxiv.org/abs/2102.12092

The PlaNet model was followed by Dreamer (Dec 2019) and DreamerV2 (Oct 2020), which train an agent using reinforcement learning using the model alone. After 200M environment steps, it surpasses Rainbow on a collection of 55 Atari games (the authors do not mention why they do not use all 57 games) when training on a single GPU for 10 days per game.

During training, a policy is learned from 486B compact states “dreamed” by the model, which is 10,000 times more than the 50M observations from the real environment (with action repeat 4).

Interestingly, the latent states are represented as a vector of several **categorical** variables – 32 variables with 32 classes each are utilized in the paper.

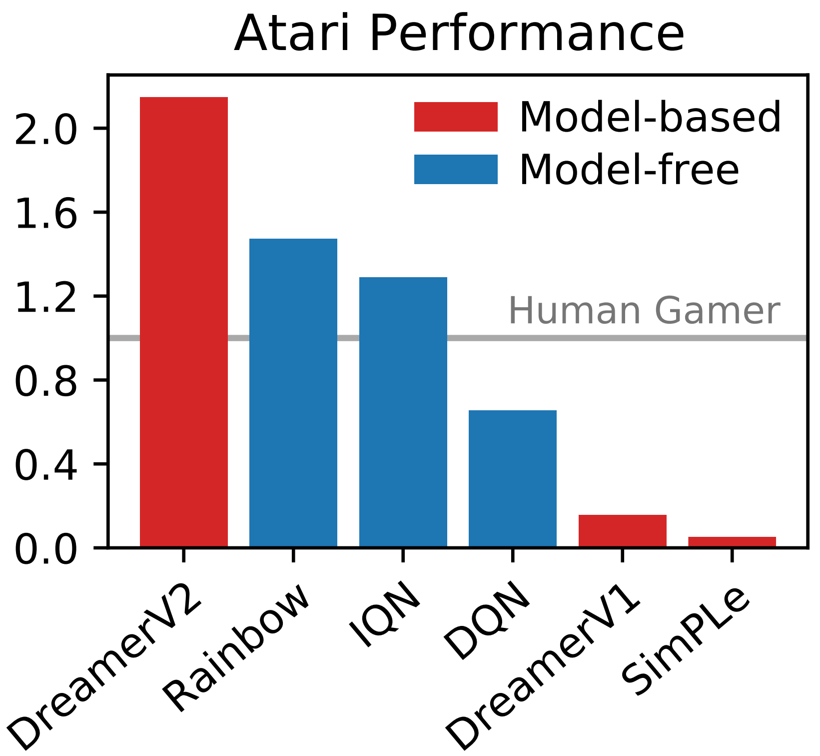


Figure 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

The model in DreamerV2 is learned using the RSSM, collecting agent experiences of observations, actions, rewards and discount factors (0.999 within episode and 0 at an episode end). Training is performed on batches of 50 sequences of length at most 50 each.

$$\begin{aligned}
\text{recurrent model:}\quad & h_t = f_\varphi(h_{t-1}, s_{t-1}, a_{t-1}),\\
\text{representation model:}\quad & s_t \sim q_\varphi(s_t | h_t, x_t),\\
\text{transition predictor:}\quad & \bar s_t \sim p_\varphi(\bar s_t | h_t),\\
\text{image predictor:}\quad & \bar x_t \sim p_\varphi(\bar x_t | h_t, s_t),\\
\text{reward predictor:}\quad & \bar r_t \sim p_\varphi(\bar r_t | h_t, s_t),\\
\text{discount predictor:}\quad & \bar\gamma_t \sim p_\varphi(\bar\gamma_t | h_t, s_t).
\end{aligned}$$

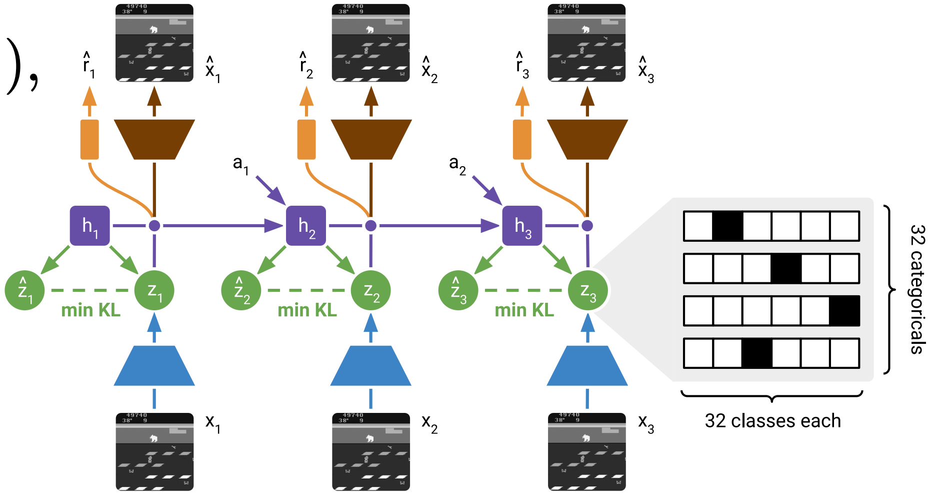


Figure 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Algorithm 1: Straight-Through Gradients with Automatic Differentiation

```
sample = one_hot(draw(logits))              # sample has no gradient
probs  = softmax(logits)                    # want gradient of this
sample = sample + probs - stop_grad(probs)  # has gradient of probs
```

Algorithm 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Model Learning

The following loss function is used:

$$\mathcal{L}(\varphi) = \mathbb{E}_{q_\varphi(s_{1:T}|a_{1:T},x_{1:T})}\Big[\sum\nolimits_{t=1}^{T} \underbrace{-\log p_\varphi(x_t|h_t,s_t)}_{\text{image log loss}} \underbrace{-\log p_\varphi(r_t|h_t,s_t)}_{\text{reward log loss}} \underbrace{-\log p_\varphi(\gamma_t|h_t,s_t)}_{\text{discount log loss}}$$
$$\underbrace{+\beta D_{\mathrm{KL}}\big[q_\varphi(s_t|h_t,x_t)\|p_\varphi(s_t|h_t)\big]}_{\text{KL loss}}\Big].$$

In the KL term, we train both the prior and the encoder. However, regularizing the encoder towards the prior makes training harder (especially at the beginning), so the authors propose **KL balancing**, minimizing the KL term faster for the prior ($\alpha = 0.8$) than for the posterior.

Algorithm 2: KL Balancing with Automatic Differentiation

```
kl_loss =       alpha  * compute_kl(stop_grad(approx_posterior), prior)
        + (1 - alpha) * compute_kl(approx_posterior, stop_grad(prior))
```

Algorithm 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Policy Learning

The policy is trained solely from the model, starting from the encountered posterior states and then considering $H = 15$ actions simulated in the compact latent state.

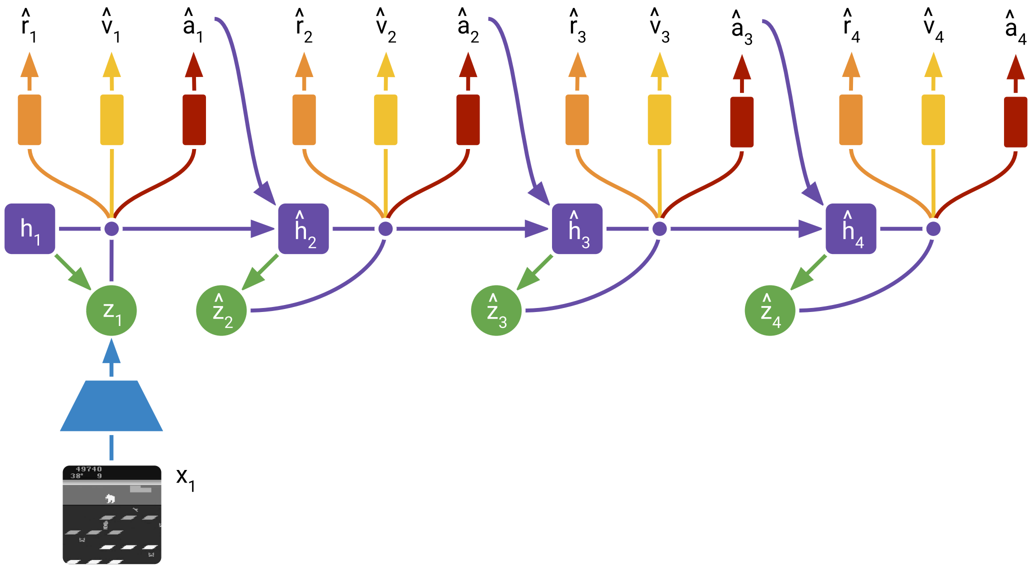


Figure 3 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

We train an actor predicting $\pi_\psi(a_t|s_t)$ and a critic predicting

$$v_\xi(s_t) = \mathbb{E}_{p_\varphi,\pi_\psi}\big[\textstyle\sum_{r\geq t}(\prod_{r'=t+1}^{r}\gamma_{r'})r_t\big].$$

The critic is trained by estimating the truncated λ-return as

$$V_t^\lambda = r_t + \gamma_t \begin{cases}(1-\lambda)v_\xi(\hat{z}_{t+1}) + \lambda V_{t+1}^\lambda & \text{if } t < H,\\ v_\xi(\hat{z}_H) & \text{if } t = H.\end{cases}$$

and then minimizing the MSE.

DreamerV2 – Policy Learning

The actor is trained using two approaches:

- the REINFORCE-like loss (with a baseline), which is unbiased, but has a high variance (even with the baseline);
- the reparametrization of discrete actions using a straight-through gradient estimation, which is biased, but has lower variance.

$$\mathcal{L}(\psi) = \mathbb{E}_{p_\varphi, \pi_\psi}\Big[\sum\nolimits_{t=1}^{H-1} \big(\underbrace{-\rho \log \pi_\psi(a_t|s_t)\,\operatorname{stopgradient}(V_t^\lambda - v_\xi(s_t))}_{\text{reinforce}} \underbrace{-(1-\rho)V_t^\lambda}_{\text{dynamics backprop}} \underbrace{-\eta H(a_t|s_t)}_{\text{entropy regularizer}}\big)\Big]$$

For Atari domains, the authors use $\rho = 0$, while for continuous actions, $\rho = 1$ works better (presumably because of the bias in case of discrete actions).

DreamerV2 – Results

The authors evaluate on 55 Atari games. They argue that the commonly used metrics have various flaws:

- **gamer-normalized median** ignores scores on half of the games,
- **gamer-normalized mean** is dominated by several games where the agent achieves super-human performance by several orders.

They therefore propose two additional ones:

- **record-normalized mean** normalizes with respect to any registered human world record for each game; however, in some games the agents still achieve super-human-record performance;
- **clipped record-normalized mean** additionally clips each score to 1; this measure is used as the primary metric in the paper.

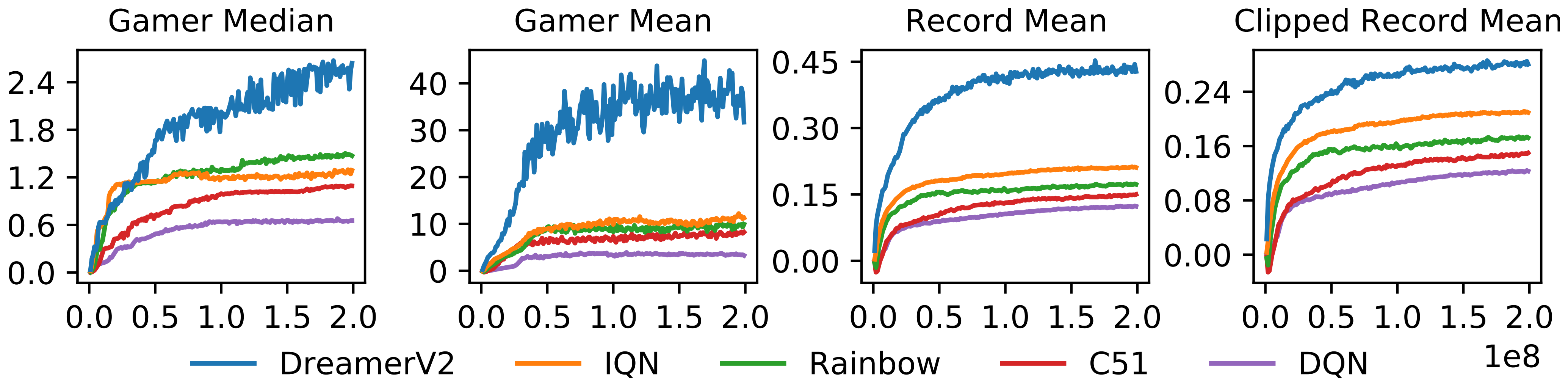


Figure 4 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Agent	Gamer Median	Gamer Mean	Record Mean	Clipped Record Mean
DreamerV2	2.15	**42.26**	**0.44**	**0.28**
DreamerV2 (schedules)	**2.64**	31.71	0.43	**0.28**
IMPALA	1.92	16.72	0.34	0.23
IQN	1.29	11.27	0.21	0.21
Rainbow	1.47	9.95	0.17	0.17
C51	1.09	8.25	0.15	0.15
DQN	0.65	3.28	0.12	0.12

Table 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Scheduling anneals actor gradient mixing ρ from 0.1 to 0, entropy loss scale, KL, lr.

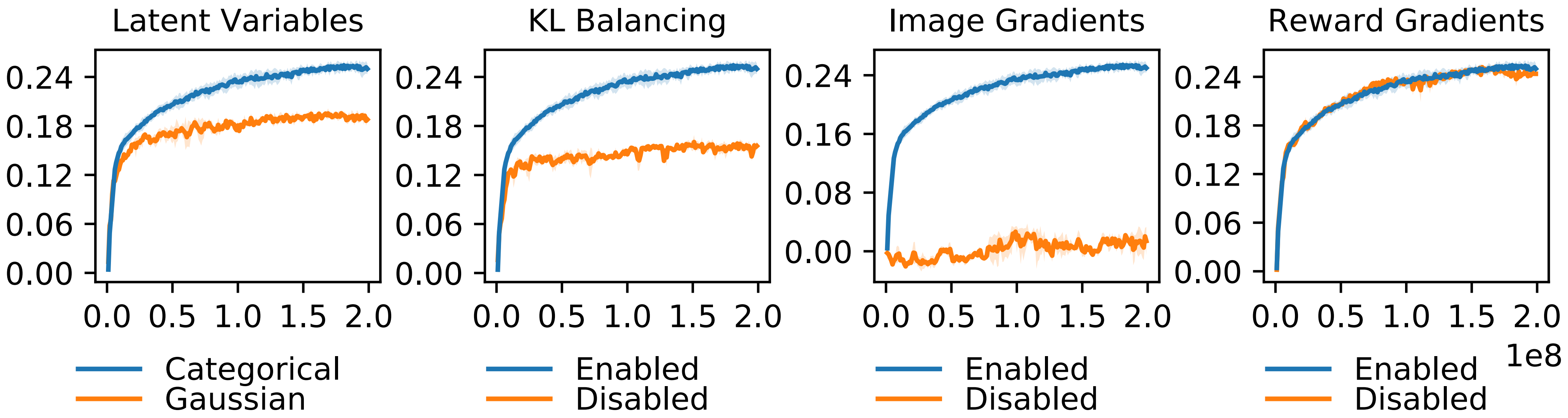


Figure 5 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Agent	Gamer Median	Gamer Mean	Record Mean	Clipped Record Mean
DreamerV2	1.64	13.39	0.36	0.25
No Layer Norm	1.66	11.29	0.38	0.25
No Reward Gradients	1.68	14.29	0.37	0.24
No Discrete Latents	0.85	3.96	0.24	0.19
No KL Balancing	0.87	4.25	0.19	0.16
No Policy Reinforce	0.72	5.10	0.16	0.15
No Image Gradients	0.05	0.37	0.01	0.01

Table 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Discrete Latent Variables

Categorical latent variables outperform Gaussian latent variables on 42 games, tie on 5 games and decrease performance on 8 games (where a tie is defined as being within 5%).

The authors provide several hypotheses why could the categorical latent variables be better:

- Categorical prior can perfectly math aggregated posterior, because mixture of categoricals is categorical, which is not true for Gaussians.
- Sparsity achieved by the 32 categorical variables with 32 classes each could be beneficial for generalization.
- Contrary to intuition, optimizing categorical variables might be easier than optimizing Gaussians, because the straight-through estimator ignores a term which would otherwise scale the gradient, which could reduce exploding/vanishing gradient problem.
- Categorical variables could be better match for modeling discrete aspect of the Atari games (defeating an enemy, collecting reward, entering a room, …).

DreamerV2 – Comparison, Hyperparametres

Algorithm	Reward Modeling	Image Modeling	Latent Transitions	Single GPU	Trainable Parameters	Atari Frames	Accelerator Days
DreamerV2	✓	✓	✓	✓	22M	200M	10
SimPLe	✓	✓	✗	✓	74M	4M	40
MuZero	✓	✗	✓	✗	40M	20B	80
MuZero Reanalyze	✓	✗	✓	✗	40M	200M	80

Table 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

World Model			Behavior			Common		
Dataset size (FIFO)	—	$2 \cdot 10^6$	Imagination horizon	H	15	Environment steps per update	—	4
Batch size	B	50	Discount	γ	0.995	MPL number of layers	—	4
Sequence length	L	50	λ-target parameter	λ	0.95	MPL number of units	—	400
Discrete latent dimensions	—	32	Actor gradient mixing	ρ	1	Gradient clipping	—	100
Discrete latent classes	—	32	Actor entropy loss scale	η	$1 \cdot 10^{-3}$	Adam epsilon	ϵ	10^{-5}
RSSM number of units	—	600	Actor learning rate	—	$4 \cdot 10^{-5}$	Weight decay (decoupled)	—	10^{-6}
KL loss scale	β	0.1	Critic learning rate	—	$1 \cdot 10^{-4}$			
KL balancing	α	0.8	Slow critic update interval	—	100			
World model learning rate	—	$2 \cdot 10^{-4}$						
Reward transformation	—	tanh						

Table D.1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

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).

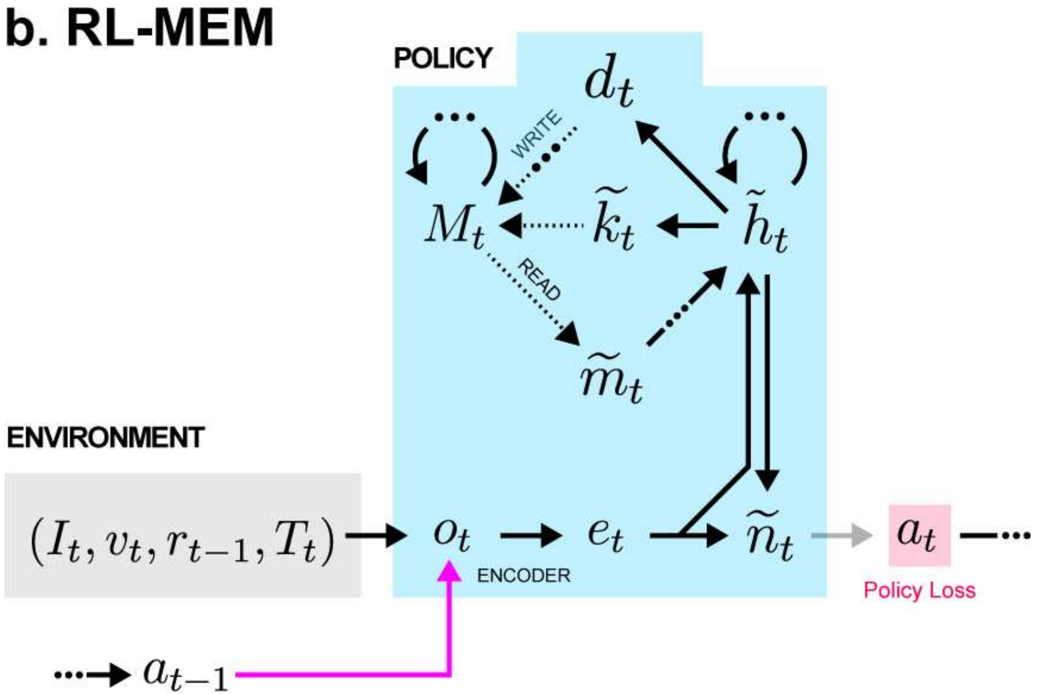


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

MERLIN – Memory Module

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

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 then 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 $2|\boldsymbol{e}|$ and K positive scalars $\beta_1, \ldots, \beta_K$.

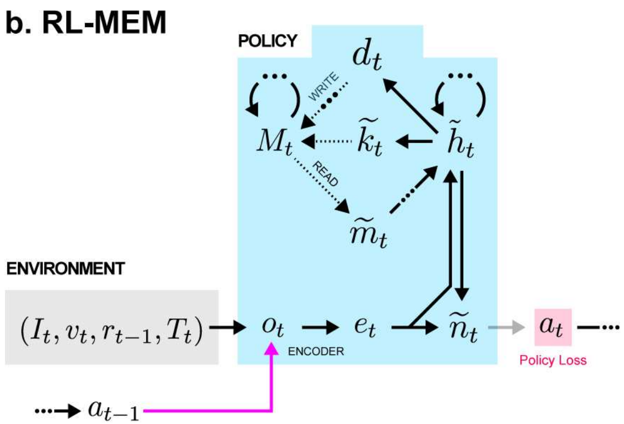


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

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 $\operatorname{softmax}$ 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 $\boldsymbol{M} \leftarrow \boldsymbol{M} + \boldsymbol{v}_{wr}[\boldsymbol{e}_t, 0] + \boldsymbol{v}_{ret}[0, \boldsymbol{e}_t]$.

However, updating the encoder and memory content purely using RL is inefficient. Therefore, MERLIN includes a *memory-based predictor (MBP)* in addition to policy. The goal of MBP is to compress observations into low-dimensional state representations $\boldsymbol{z}$ 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 $\boldsymbol{z}_t$ predicts next state variable conditioned on history of state variables and actions $p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}, a_{t-1}, \ldots, \boldsymbol{z}_1, a_1)$, and *posterior* corrects the prior using the new observation $\boldsymbol{o}_t$, forming a better estimate $q(\boldsymbol{z}_t | \boldsymbol{o}_t, \boldsymbol{z}_{t-1}, a_{t-1}, \ldots, \boldsymbol{z}_1, a_1)$.

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.

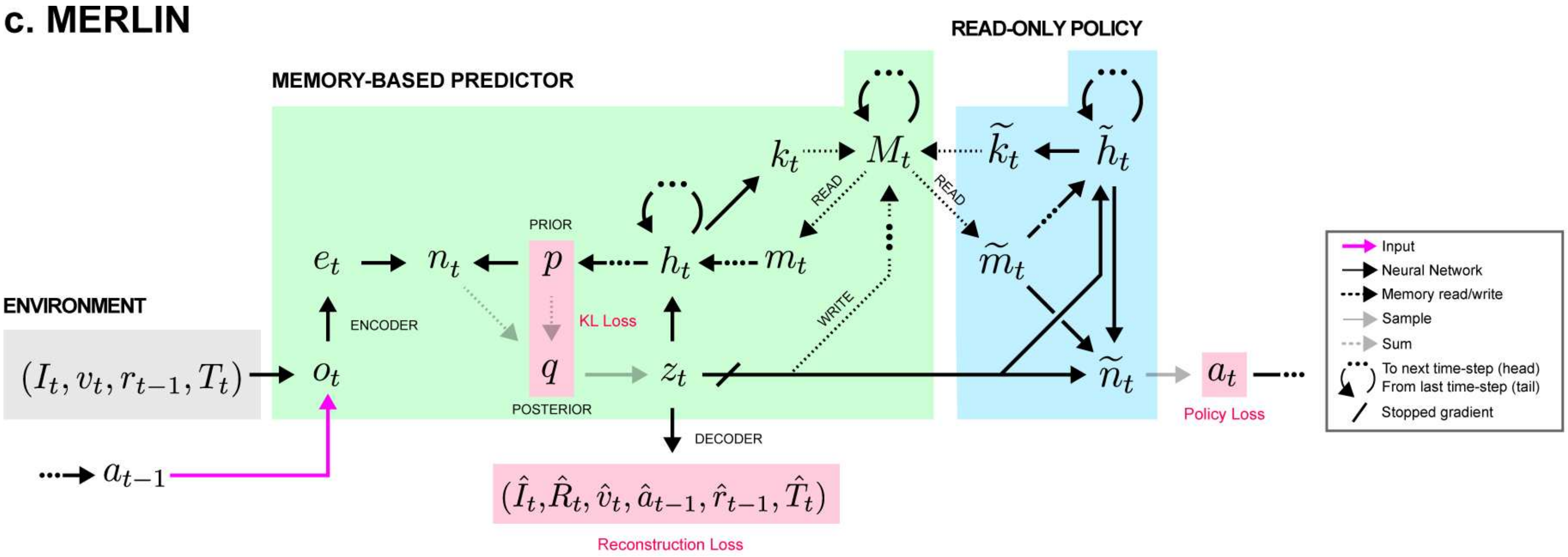


Figure 1c of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

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 $T > T_{\max}$

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

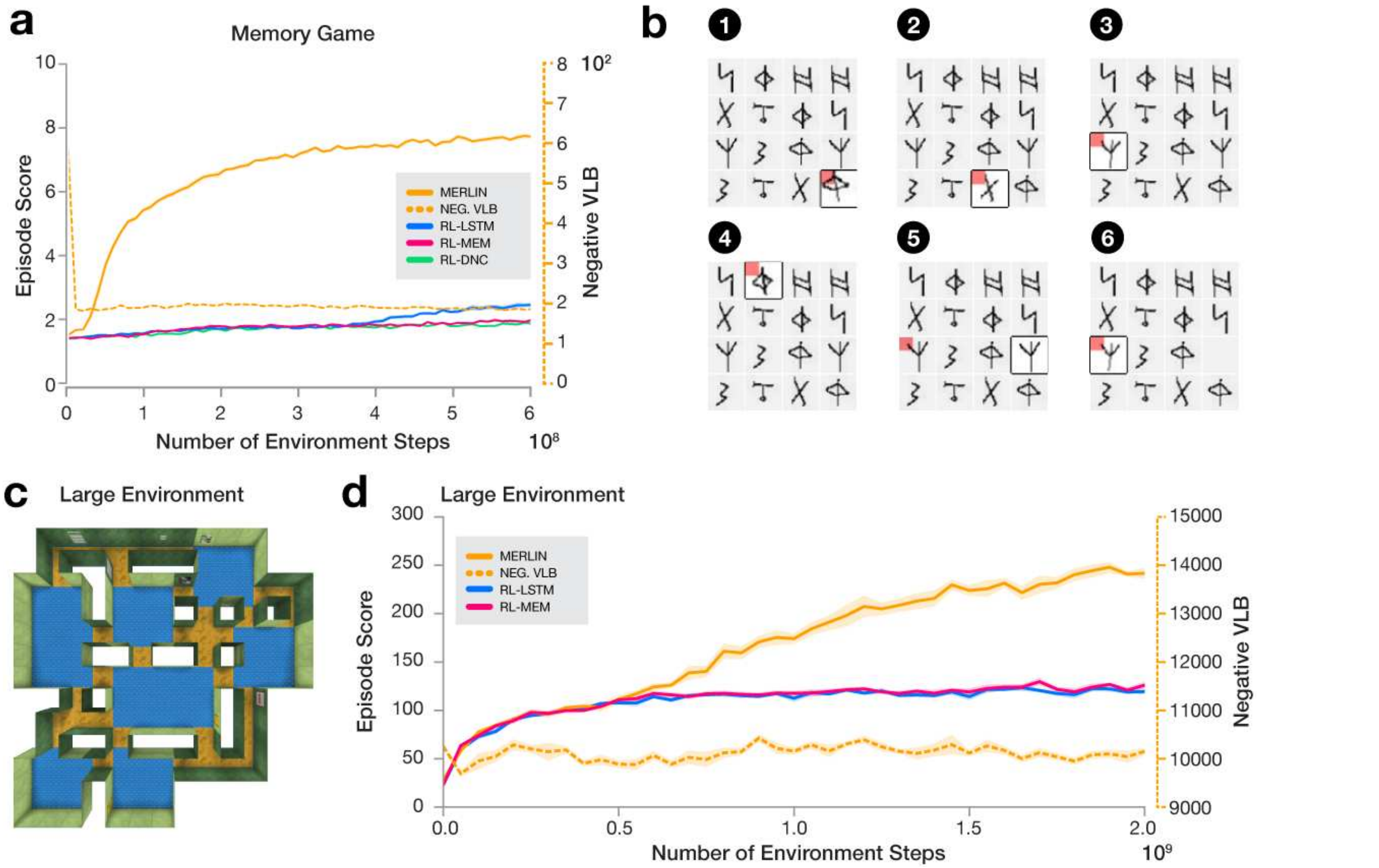


Figure 2 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

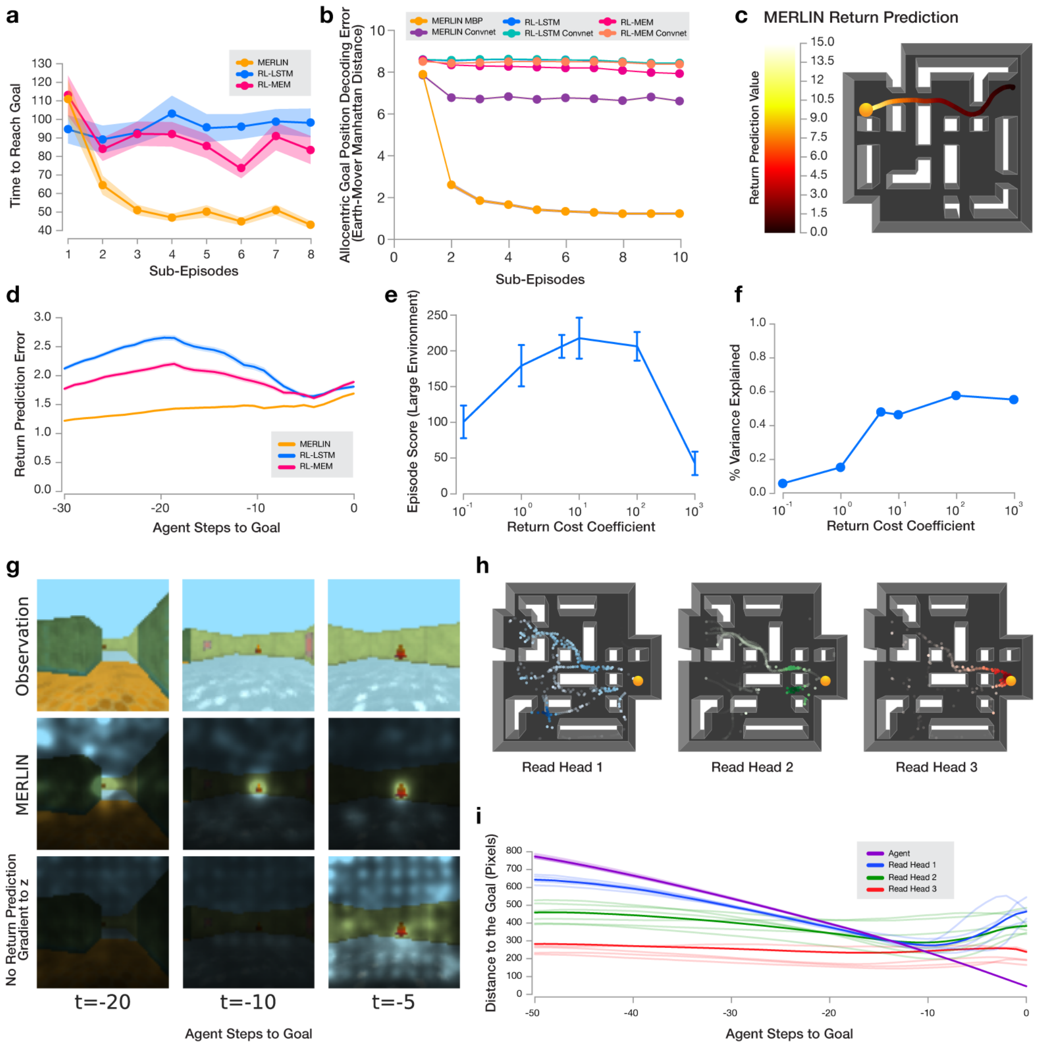
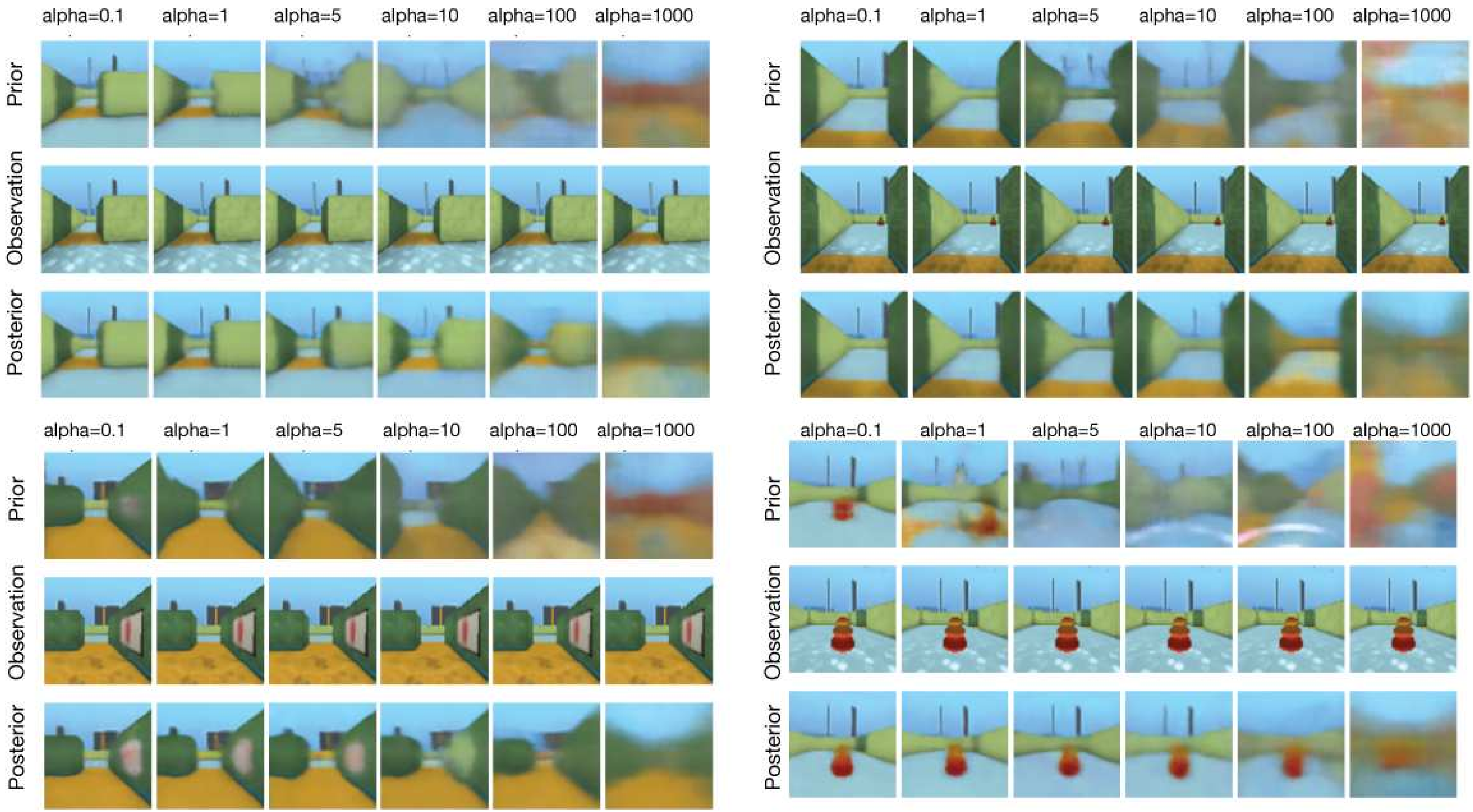


Figure 3 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.



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

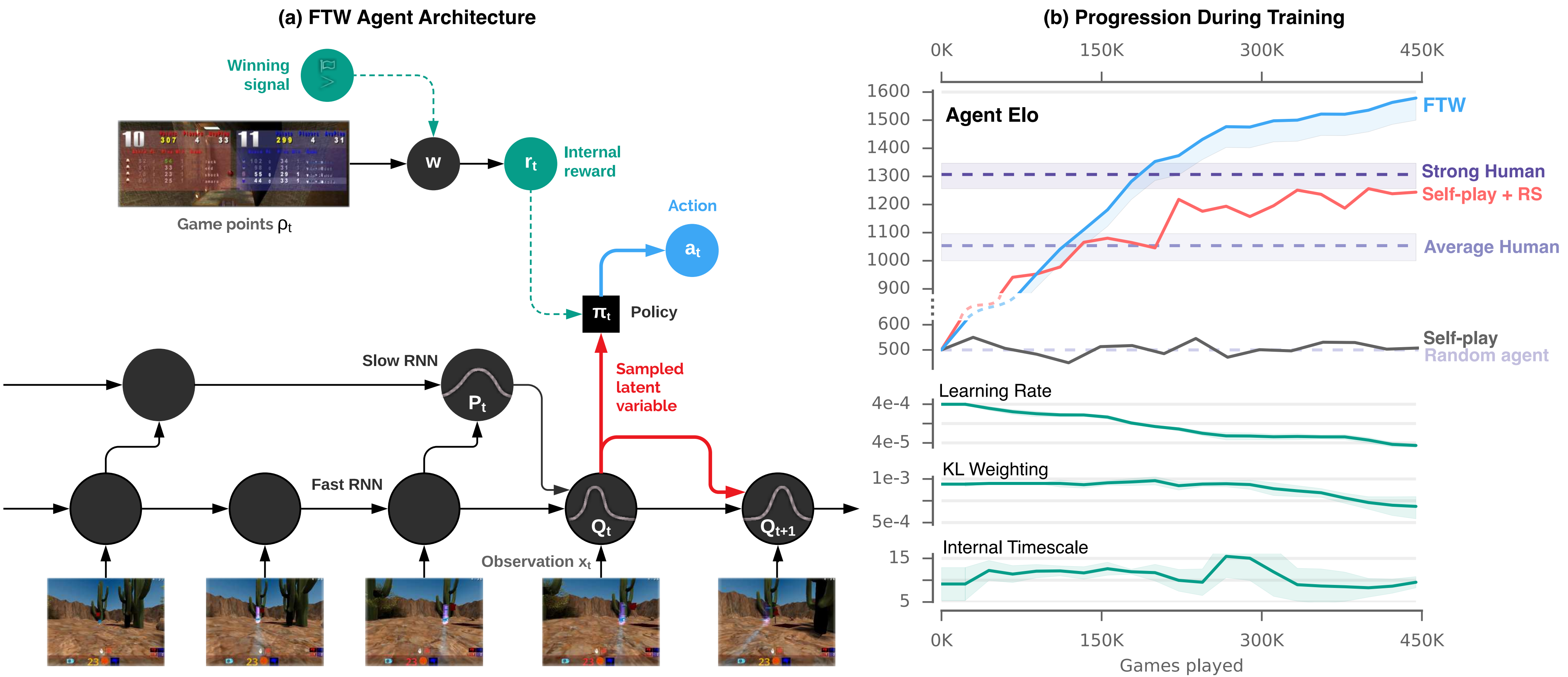


Figure 2 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

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

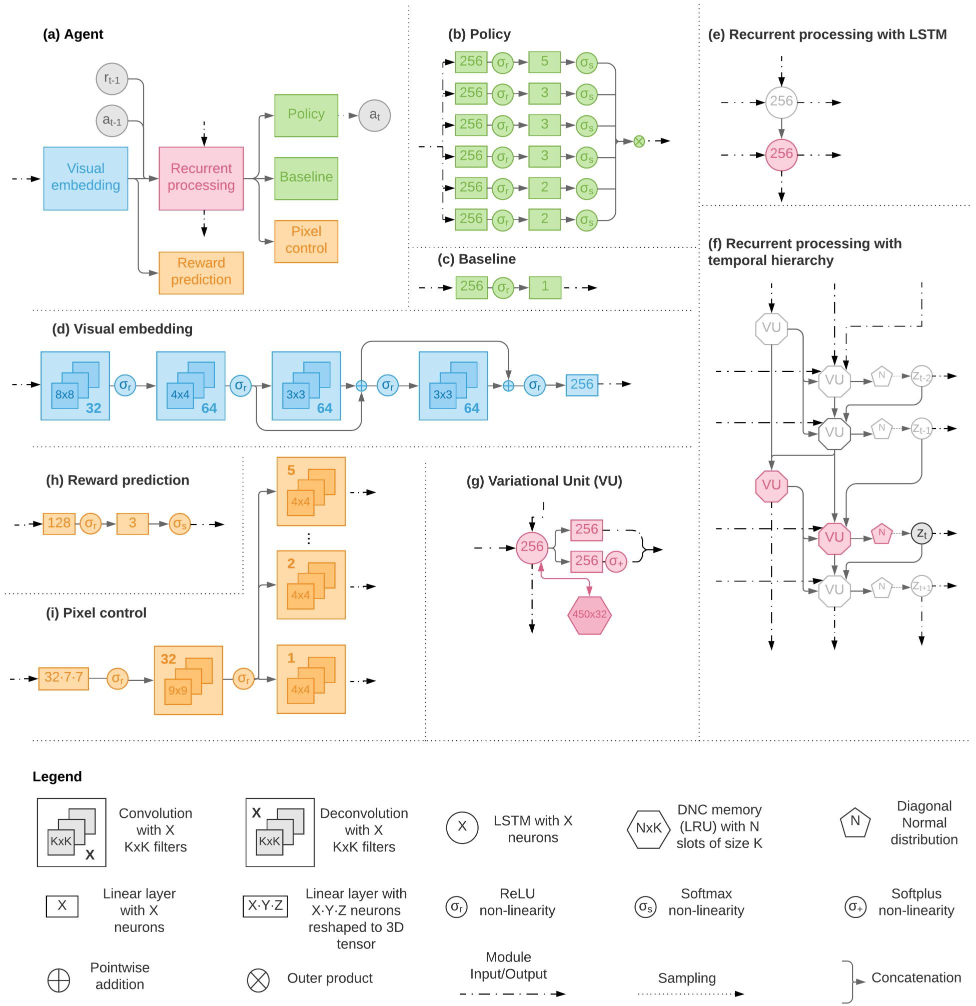


Figure S10 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

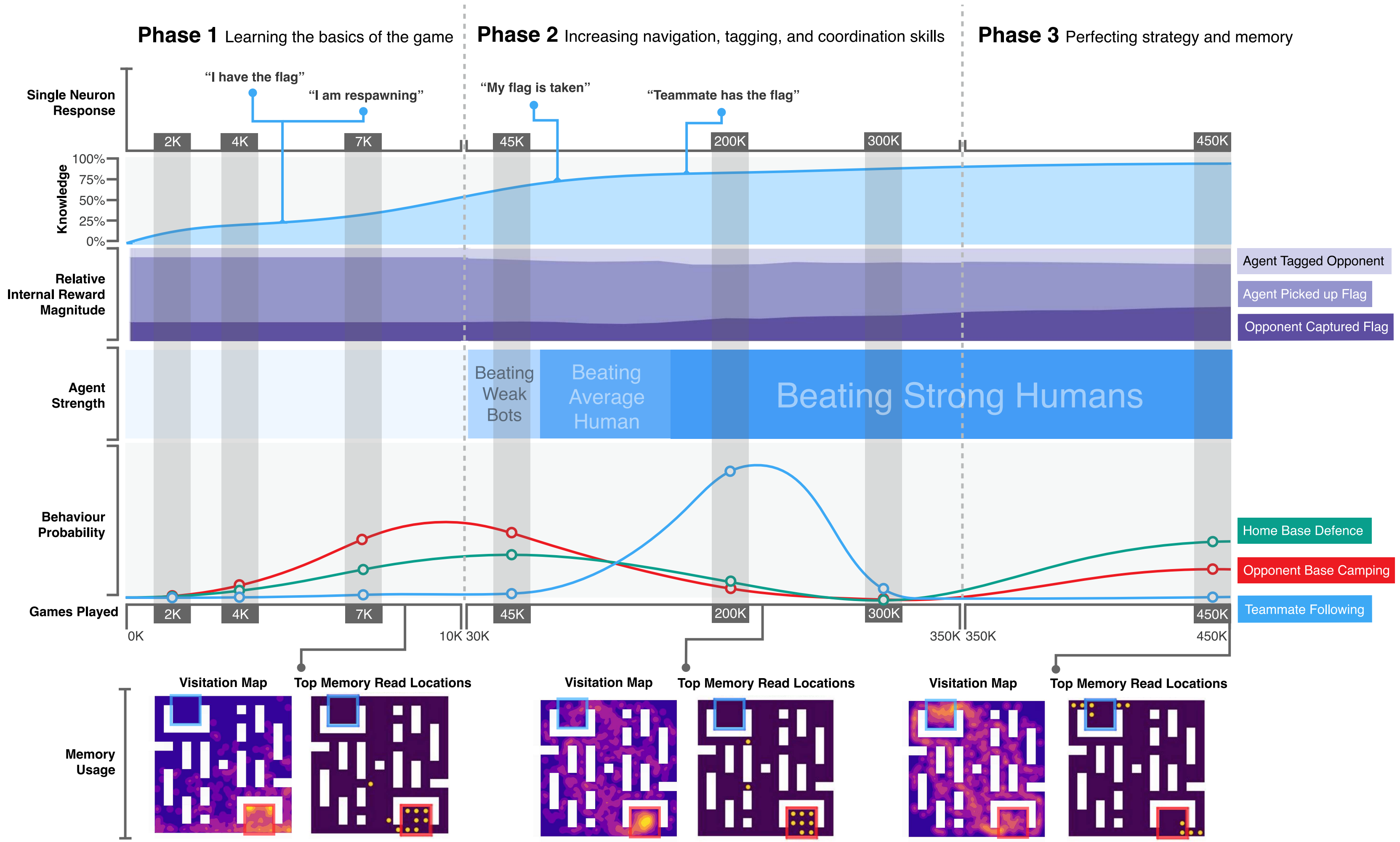


Figure 4 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.