

REINFORCEMENT LEARNING WITH LATENT FLOW

Anonymous authors

Paper under double-blind review

ABSTRACT

Temporal information is essential to learning effective policies with Reinforcement Learning (RL). However, current state-of-the-art RL algorithms either assume that such information is given as part of the state space or, when learning from pixels, use the simple heuristic of frame-stacking to implicitly capture temporal information present in the image observations. This heuristic is in contrast to the current paradigm in video classification architectures, which utilize explicit encodings of temporal information through methods such as optical flow and two-stream architectures to achieve state-of-the-art performance. Inspired by leading video classification architectures, we introduce the **Flow of Latents for Reinforcement Learning (Flare)**, a network architecture for RL that explicitly encodes temporal information through latent vector differences. We show that Flare (i) recovers optimal performance in state-based RL without explicit access to the state velocity, solely with positional state information, (ii) achieves state-of-the-art performance on pixel-based continuous control tasks within the DeepMind control benchmark suite, (iii) is the most sample efficient model-free pixel-based RL algorithm on challenging environments in the DeepMind control suite such as quadruped walk, hopper hop, finger turn hard, pendulum swing, and walker run, outperforming the prior model-free state-of-the-art by $1.9\times$ and $1.5\times$ on the 500k and 1M step benchmarks, respectively, and (iv), **when augmented over rainbow DQN, outperforms or matches the baseline on a diversity of challenging Atari games at 50M time step benchmark.**

1 INTRODUCTION

Reinforcement learning (RL) (Sutton & Barto, 1998) holds the promise of enabling artificial agents to solve a diverse set of tasks in uncertain and unstructured environments. Recent developments in RL with deep neural networks have led to tremendous advances in autonomous decision making. Notable examples include classical board games (Silver et al., 2016; 2017), video games (Mnih et al., 2015; Berner et al., 2019; Vinyals et al., 2019), and continuous control (Schulman et al., 2017; Lillicrap et al., 2016; Rajeswaran et al., 2018). A large body of research has focused on the case where an RL agent is equipped with a compact state representation. Such compact state representations are typically available in simulation (Todorov et al., 2012; Tassa et al., 2018) or in laboratories equipped with elaborate motion capture systems (OpenAI et al., 2018; Zhu et al., 2019; Lowrey et al., 2018). However, state representations are seldom available in unstructured real-world settings like the home. For RL agents to be truly autonomous and widely applicable, sample efficiency and the ability to act using raw sensory observations like pixels is crucial. Motivated by this understanding, we study the problem of efficient and effective deep RL from pixels.

A number of recent works have made progress towards closing the sample-efficiency and performance gap between deep RL from states and pixels (Laskin et al., 2020b;a; Hafner et al., 2019a; Kostrikov et al., 2020). An important component in this endeavor has been the extraction of high quality visual features during the RL process. Laskin et al. (2020a) and Stooke et al. (2020) have shown that features learned either explicitly with auxiliary losses (reconstruction or contrastive losses) or implicitly (through data augmentation) are sufficiently informative to recover the agent’s pose information. While existing methods can encode positional information from images, there has been little attention devoted to extracting temporal information from a stream of images. As a result, existing deep RL methods from pixels struggle to learn effective policies on more challenging continuous control environments that deal with partial observability, sparse rewards, or those that require precise manipulation.

A APPENDIX

A.1 SOLVED DMCONTROL ENVIRONMENTS

We show in Figure 9 that Flare matches the model-free state-of-the-art on the six most commonly benchmarked DMControl environments in Laskin et al. (2020a); Kostrikov et al. (2020); Hafner et al. (2019a). These environments are considered simple because the state-of-the-art pixel-based SAC variants are as efficient as state-based SAC, suggesting that the pixel-based SAC performance is effectively saturated. For this reason, we benchmark against more challenging environments in the main text.

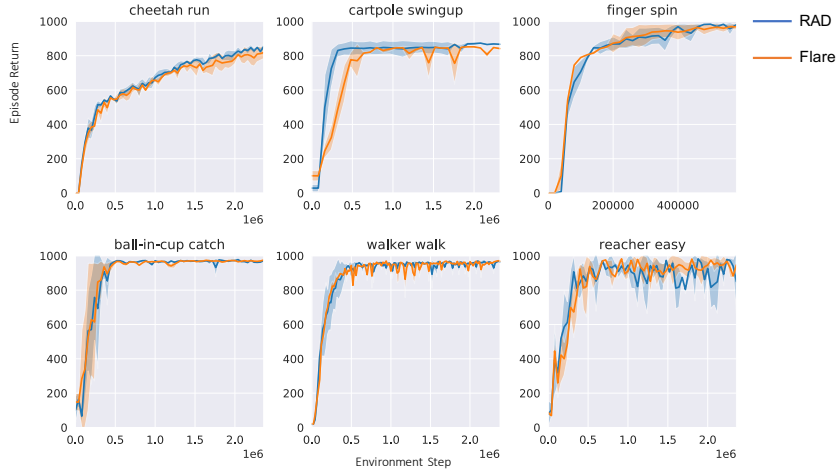


Figure 9: Pixel-based RAD and Flare on classic environments: We compare Flare to RAD on six classic environment that can already be solved by RAD. Flare matches the performance with RAD on all of them.

A.2 COMPARE FLARE WITH DQN VARIANTS ON ATARI

	Flare	Rainbow	DQN	Prioritized	QR DQN	IQN	double Q
Assault	9466±1928	10123±2061	1475±215	3016±802	10215±1255	12211 ±2434	1904±271
Breakout	330±10	321±34	283±50	361±33	392±15	433 ±65	333±32
Freeway	34 ±0	34 ±0	24±12	25±12	33±1	33±0	25±12
Krull	8423±173	8030±717	6480±996	8600±194	10283 ±751	9510±459	8088±411
Montezuma	400 ±0	0±0	0±0	0±0	0±0	0±0	0±0
Seaquest	8362 ±1180	4521±3554	2330±847	5628±780	5564±835	15015±3734	7531±1116
Up n Down	44055 ±12746	24568±2216	5246±3416	11693±4635	17148±5897	29973±20263	7421±2860
Tutankham	240 ±7	148±16	79±21	158±21	210±17	180±13	122±24

Table 3: Evaluation on 8 benchmark Atari games at 50M training steps. Flare and Rainbow are over 3 seeds. The rest are over 5 seeds, directly taken from DQN Zoo repository.

A.3 IMPLEMENTATION DETAILS FOR PIXEL-BASED EXPERIMENTS

Hyperparameter	Value
Augmentation	Random Translate
Observation size	(100, 100)
Augmented size	(108, 108)
Replay buffer size	100000
Initial steps	10000
Training environment steps	1.5e6 pendulum swingup 2.5e6 others
Batch size	128
Stacked frames	2 pendulum swingup 3 others
Action repeat	2 walker run, hopper hop 4 others
Camera id	2 quadruped, walk 0 others
Evaluation episode length	10
Hidden units (MLP)	1024
Number of layers (MLP)	2
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_{CNN}, \pi_\psi, Q_\phi)$	(.9, .999)
$(\beta_1, \beta_2) \rightarrow (\alpha)$	(.5, .999)
Learning Rate (π_ψ, Q_ϕ)	$2e-4$
Learning Rate (f_{CNN})	$1e-3$
Learning Rate (α)	$1e-4$
Critic target update frequency	2
Critic EMA τ	0.01
Encoder EMA τ	0.05
Convolutional layers	4
Number of CNN filters	32
Latent dimension	64
Non-linearity	ReLU
Discount γ	0.99
Initial Temperature	0.1

A.4 IMPLEMENTATION DETAILS FOR STATE-BASED MOTIVATION EXPERIMENTS

Hyperparameter	Value
Replay buffer size	2000000
Initial steps	5000
Batch size	1024
Stacked frames	4 Flare, Stack SAC; 1 otherwise
Action repeat	1
Evaluation episode length	10
Hidden units	1024
Number of layers	2
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_{CNN}, \pi_\psi, Q_\phi)$	(.9, .999)
$(\beta_1, \beta_2) \rightarrow (\alpha)$	(.9, .999)
Learning Rate (π_ψ, Q_ϕ)	$1e-4$
Learning Rate (α)	$1e-4$
Critic target update frequency	2
Critic EMA τ	0.01
Non-linearity	ReLU
Discount γ	0.99
Initial Temperature	0.1

A.5 INTERPRETING FLARE FROM A TWO-STREAM PROSPECTIVE

Let f_{CNN} and o'_t be the pixel encoder and the augmented observation in Flare. Then, $z_t = f_{\text{CNN}}(o'_t)$ denotes the latent encoding for a frame at time t . By computing the latent flow $\delta_t = z_t - z_{t-1}$, Flare essentially approximates $\frac{\partial z_t}{\partial t}$ via backward finite difference. Then following chain rule, we have

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial o'} \cdot \frac{\partial o'}{\partial t} = \frac{f_{\text{CNN}}'(o')}{\partial o'} \cdot \text{dense optical flow}$$

indicating that Flare eventually uses dense optical flow by propagating it through the derivative of the trained encoder. While the two-stream architecture trains a spatial stream CNN from RGB channels and a temporal stream from optical flow separately, Flare can be interpreted as training one spatial stream encoder from the RL objective and approximate the temporal stream encoder with its derivative.