DeepMind

Reasoning-Modulated Representations

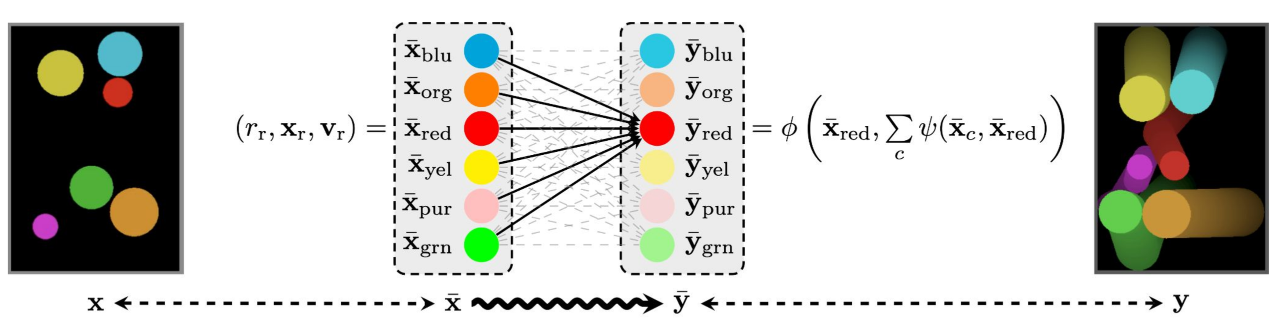
Petar Veličković*, Matko Bošnjak*, Thomas Kipf, Alexander Lerchner, Raia Hadsell, Razvan Pascanu, Charles Blundell

TL;DR;

By incorporating information about the generative process of our task into a pre-trained reasoning module, we learn better representations in a self-supervised learning settings from pixels.

Motivation

Predicting the next raw pixel output $\mathbf{y}$ from a raw pixel input $\mathbf{x}$ is hard. If we were to do that for a system of bouncing balls, knowing that an algorithm underlies this task should help us in some way. With a suitably abstractified data $\bar{\mathbf{x}}$, predicting the future abstract state $\bar{\mathbf{y}}$ could be as easy as running a force calculation algorithm.



Though this abstraction seems to simplify the path from $\mathbf{x}$ to $\mathbf{y}$, it convolutes our efforts as now we need to take care of a bigger pipeline $\mathbf{x} \to \bar{\mathbf{x}} \rightsquigarrow \bar{\mathbf{y}} \to \mathbf{y}$ where

$\mathbf{x} \to \bar{\mathbf{x}}$ requires the knowledge of right abstraction or a massive paired dataset to learn the mapping

$\bar{\mathbf{x}} \rightsquigarrow \bar{\mathbf{y}}$ implies a perfect algorithm, which in reality we might not have

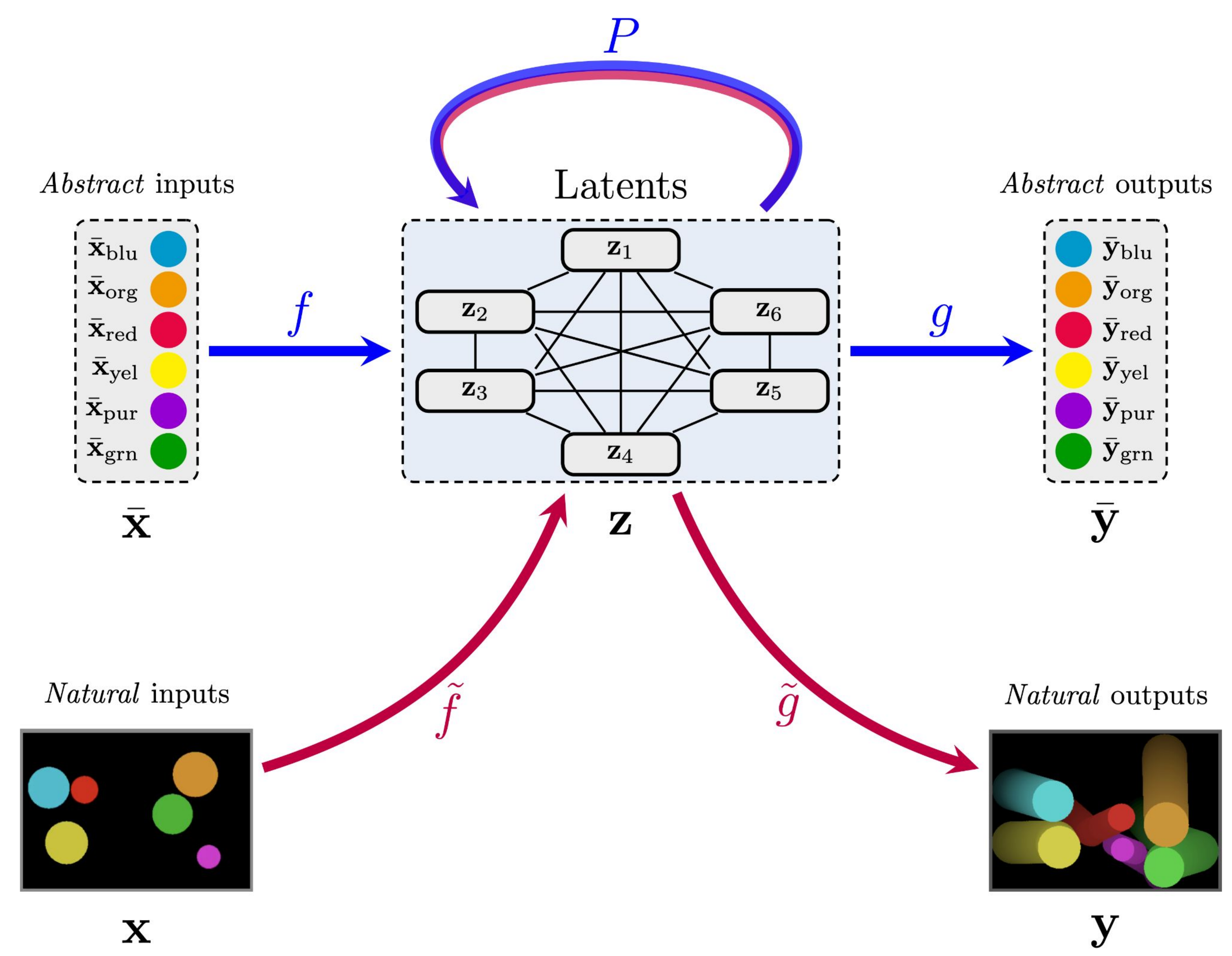
$\bar{\mathbf{y}} \to \mathbf{y}$ calls for a differentiable renderer or a massive paired dataset to learn the mapping

Architecture

Two-stage encode-process-decode

1st stage: train the $\bar{\mathbf{x}} \rightsquigarrow \bar{\mathbf{y}}$ pathway with the encode-process-decode architecture $\bar{\mathbf{x}} \xrightarrow{f} \mathbf{z} \xrightarrow{P} \mathbf{z}' \xrightarrow{g} \bar{\mathbf{y}}$
- encoder f : learns to map abstract inputs $\bar{\mathbf{x}}$ into a high-dimensional latent $\mathbf{z}$
- processor P : learns a "neural executor" in a high-dimensional space
- decoder g : learns to map the high-dimensional latent $\mathbf{z}'$ into the abstract output $\bar{\mathbf{y}}$

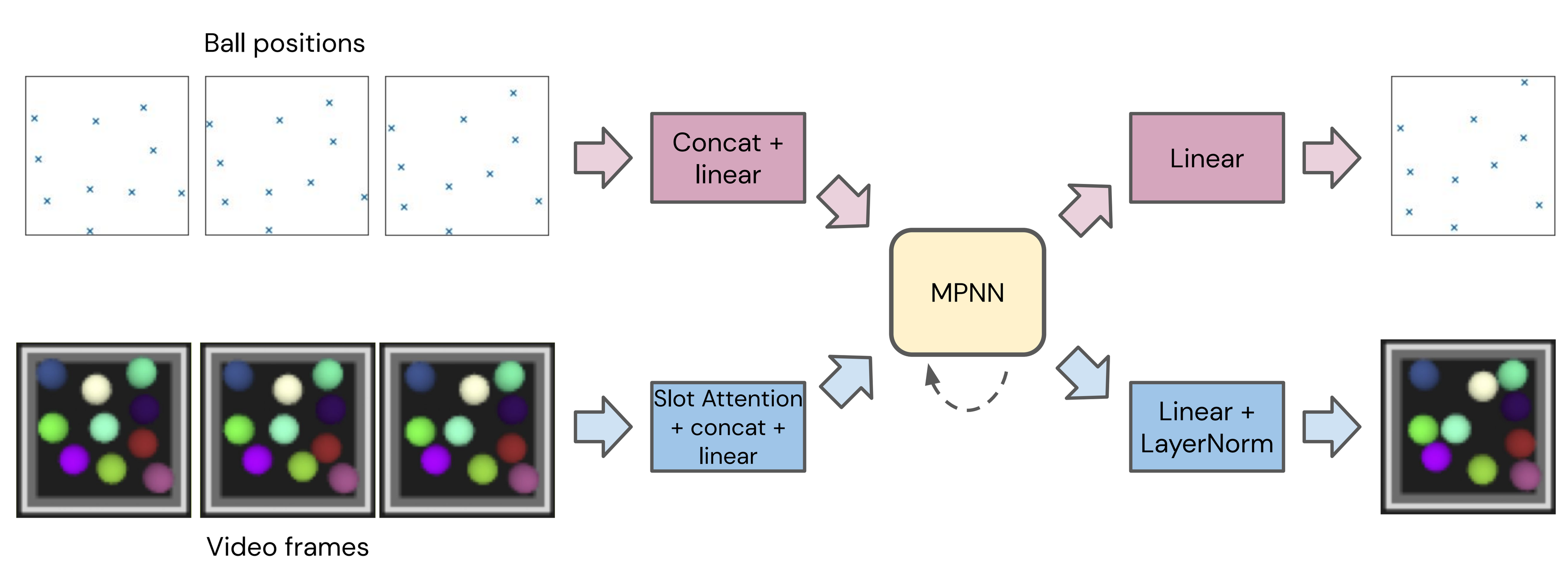
P is now a differentiable module that learned to simulate $\bar{\mathbf{x}} \rightsquigarrow \bar{\mathbf{y}}$ in a high-dimensional space



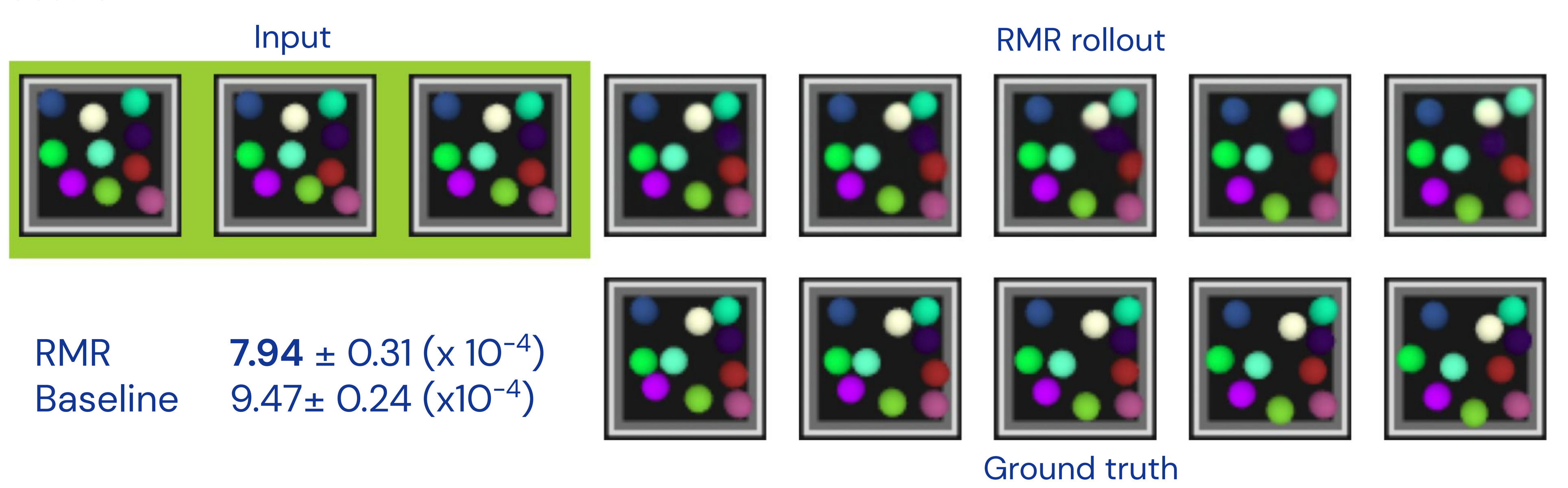
2nd stage: train the $\mathbf{x} \to \mathbf{y}$ pathway with the encode-process-decode architecture $\mathbf{x} \xrightarrow{\tilde{f}} \mathbf{z} \xrightarrow{P} \mathbf{z}' \xrightarrow{\tilde{g}} \mathbf{y}$ where we swapped out abstract encoders and decoders for natural ones
- encoder $\tilde{f}$: learns to map pixel inputs $\mathbf{x}$ into the high-dimensional latent $\mathbf{z}$
- processor P : *frozen* from the previous step to retain the semantics of its mapping
- decoder $\tilde{g}$: learns to map the high-dimensional latent $\mathbf{z}'$ into the pixel output $\mathbf{y}$

Bouncing balls

Training architecture



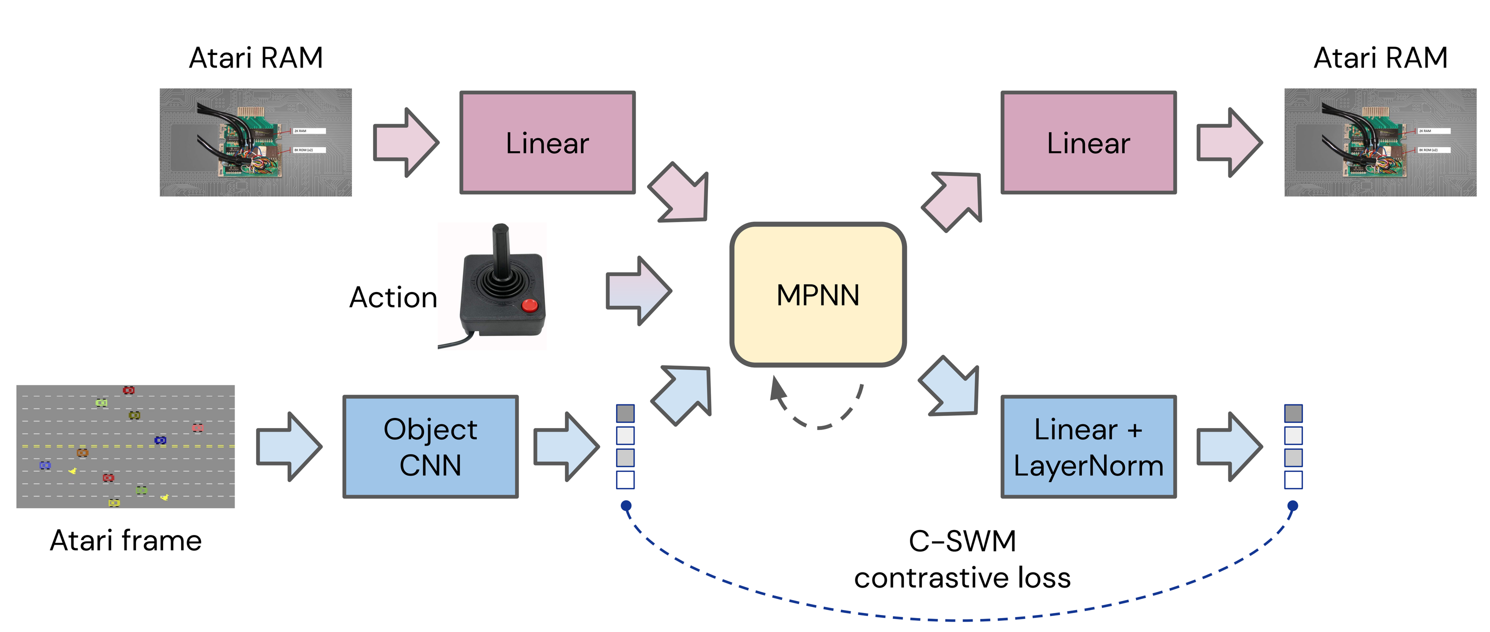
Results



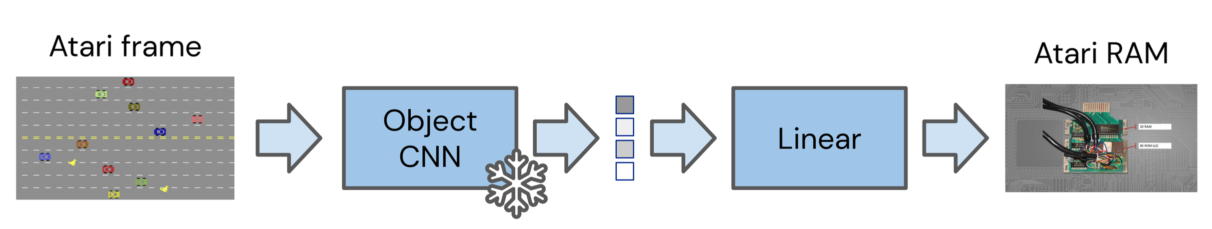
RMR	**7.94** ± 0.31 (x 10^{-4})
Baseline	9.47± 0.24 (x10^{-4})

Atari

Training architecture



Evaluating representation quality



Results

- Significantly better than C-SWM on 12 / 19 games
- Indistinguishable on 7 / 19

Table 1. Natural modelling results for Atari 2600. Bit-level F_1 reported for slots with high entropy, as in (Anand et al., 2019). Results are considered **significant** at $p < 0.05$ (paired t-test).

Game	C-SWM	RMR	p-value
Asteroids	0.597±0.002	**0.602**±0.003	**0.006**
Berzerk	0.533±0.022	0.528±0.033	0.368
Bowling	0.949±0.003	0.951±0.002	0.110
Boxing	0.667±0.011	**0.678**±0.006	**0.040**
Breakout	0.839±0.014	**0.868**±0.003	**0.002**
Freeway	0.917±0.018	**0.938**±0.003	**0.020**
Frostbite	0.596±0.020	**0.641**±0.008	**0.004**
H.E.R.O.	0.799±0.016	**0.845**±0.016	**0.004**
Montezuma	0.829±0.006	0.829±0.023	0.490
Ms. Pac-Man	0.606±0.005	0.604±0.003	0.246
Pitfall!	0.608±0.008	**0.633**±0.016	**0.012**
Pong	0.765±0.009	**0.774**±0.004	**0.025**
Private Eye	0.859±0.009	**0.874**±0.007	**0.043**
River Raid	0.764±0.003	**0.771**±0.002	**0.008**
Skiing	0.770±0.009	0.769±0.017	0.345
Space Invaders	0.779±0.004	0.779±0.003	0.363
Tennis	0.728±0.003	**0.735**±0.002	**0.004**
Venture	0.637±0.005	0.639±0.002	0.337
Yars' Revenge	0.772±0.002	**0.778**±0.001	**0.002**