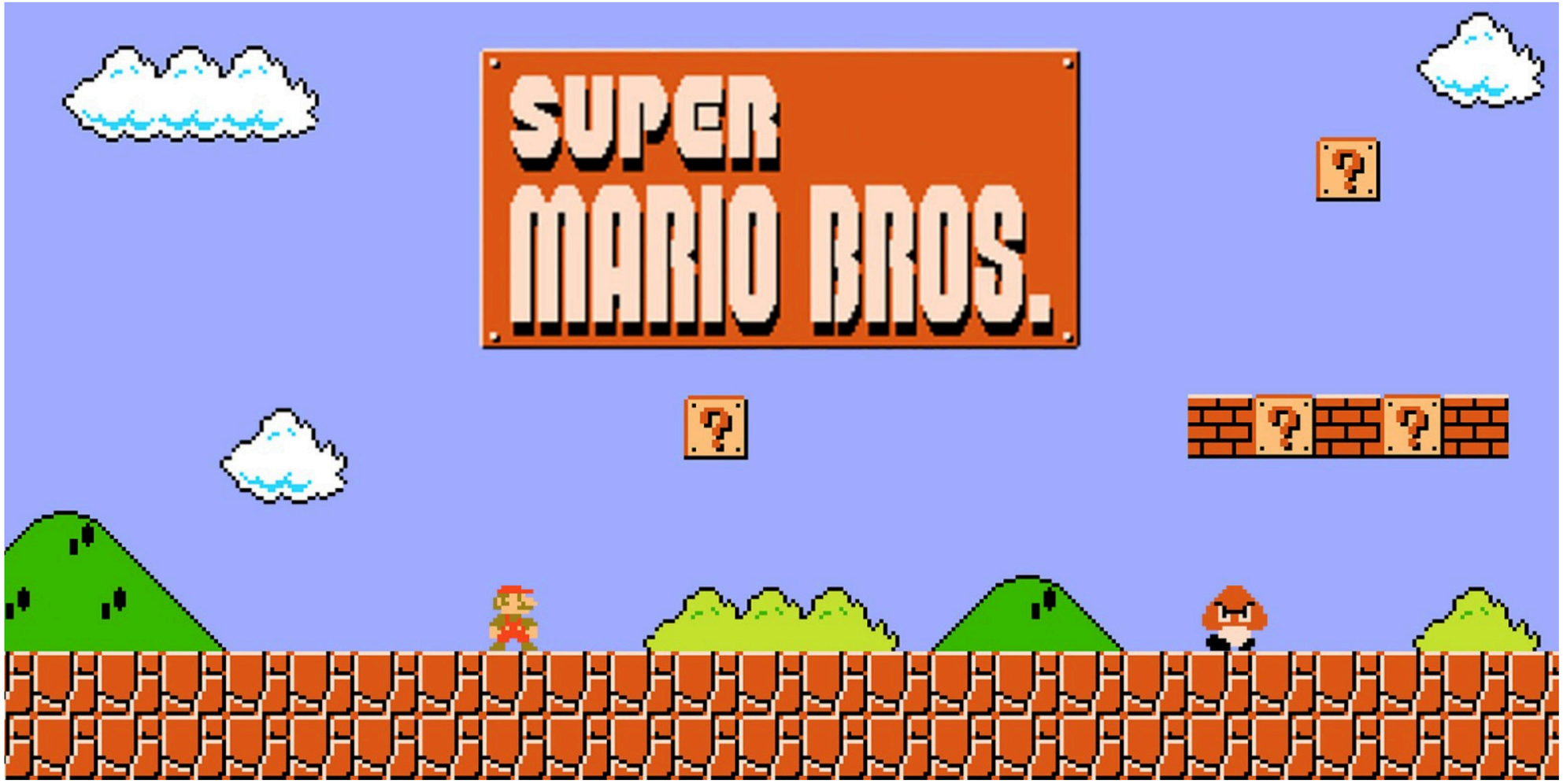
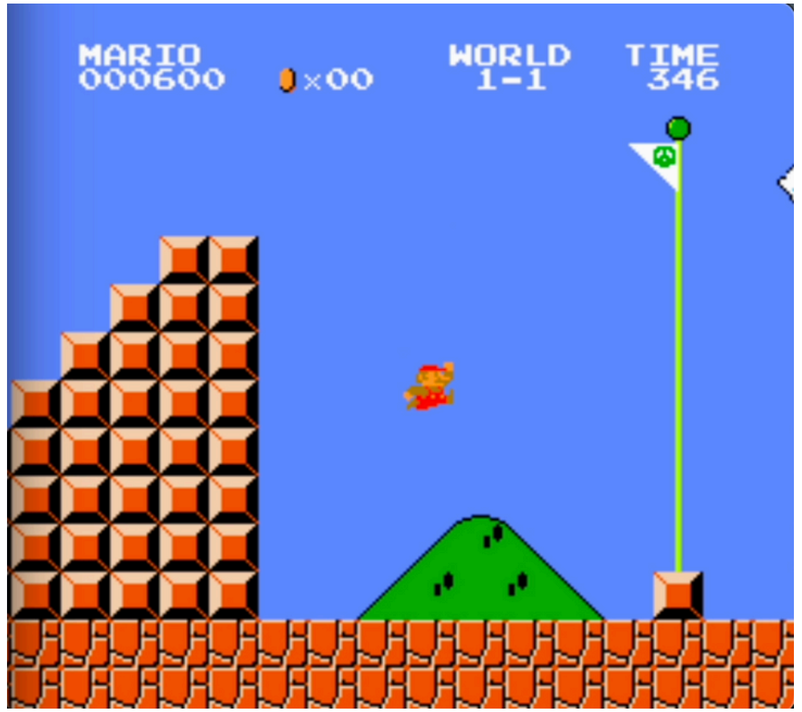
Reinforcement Learning Project



Marco Barrasso, Edoardo Insaghi
Università degli Studi di Trieste, 2024

Goals of the project

Train a deep reinforcement learning agent able to finish the first level of Super Mario Bros in an environment with dense rewards



*Curiosity-driven Exploration by Self-supervised Prediction

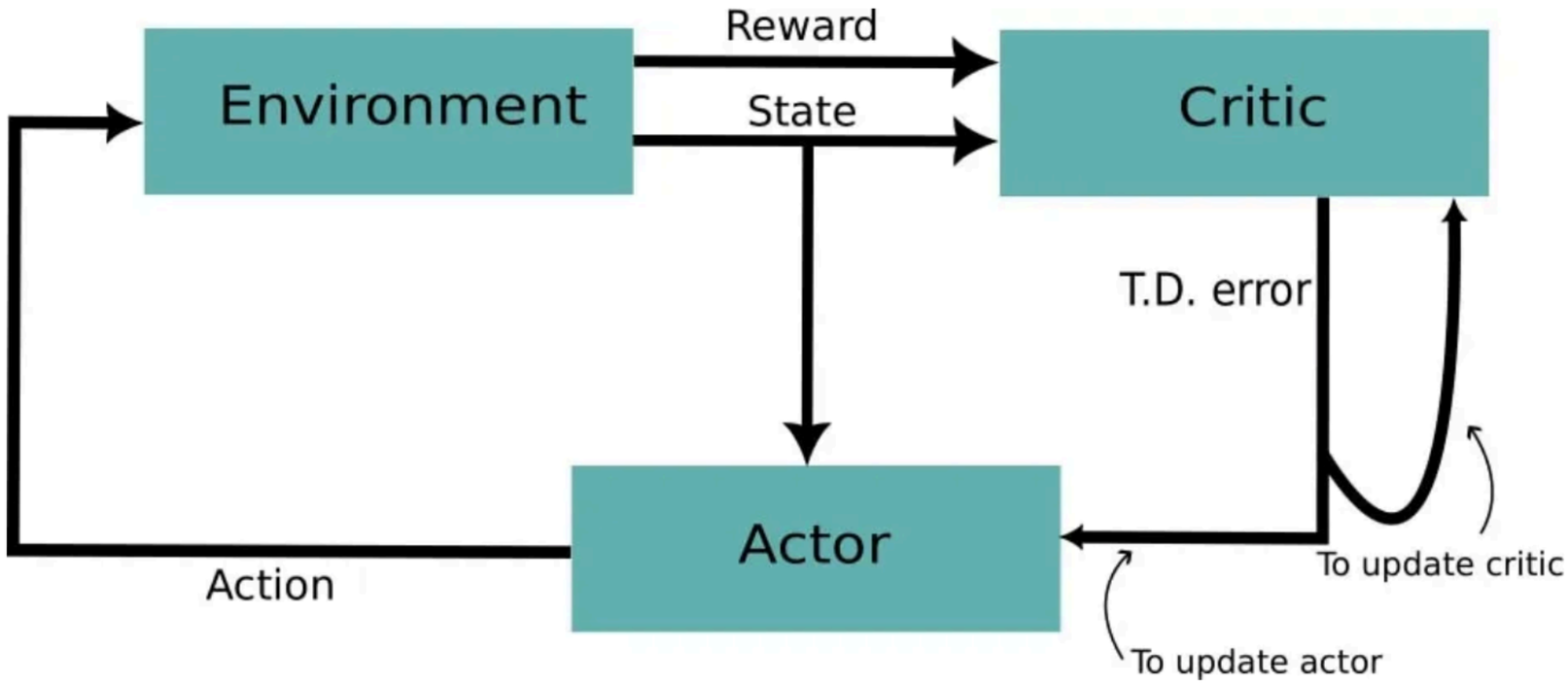
(a) learn to explore in Level-1 (b) explore faster in Level-2

Figure 1. Discovering how to play *Super Mario Bros* **without rewards**. (a) Using only curiosity-driven exploration, the agent makes significant progress in Level-1. (b) The gained knowledge helps the agent explore subsequent levels much faster than when starting from scratch. Watch the video at http://pathak22.github.io/noreward-rl/

Implement the Intrinsic Curiosity Module described in the paper* and evaluate the performance of the agent in a context of extremely sparse rewards

Actor Critic - Policy Gradient

On policy RL algorithm that learns a stochastic policy combining two components: the actor $\pi[a_t|\mathbf{s}_t, \boldsymbol{\theta}]$ which decides which action to take given the state the agent is in, and the critic $v[\mathbf{s}_t, \boldsymbol{\phi}]$ which evaluates how good an action was by estimating the value of the state reached with it



A3C - Pseudo code

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

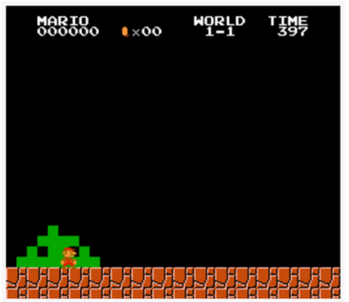
// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$ - Entropy Loss
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

The Open-AI Gym Environment

Environment	Game	ROM	Screenshot
`SuperMarioBros-v0`	SMB	standard	

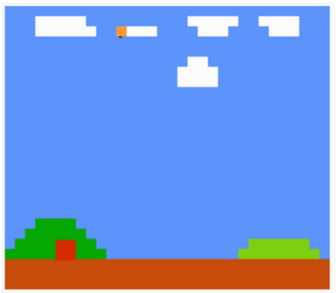
We chose the original version of the game, but the environment offers more options:

- downsample



- pixel

- rectangle



Additional Information provided by the step() function of the environment:

- coins collected
- how many lifes are left
- agent collected the flag
- score (!= reward)
- stage
- status of the agent
- time left
- world
- x_pos
- y_pos

The Open-AI Gym Environment

How is the reward function defined?

- Small negative reward each second
- Big positive reward for finishing the level
- Big negative reward for dying
- Positive reward for going right
- Negative reward for going left

Calculated as: $x\_pos_{t+1} - x\_pos_t$

+ Scaling for faster neural net training

Action Space

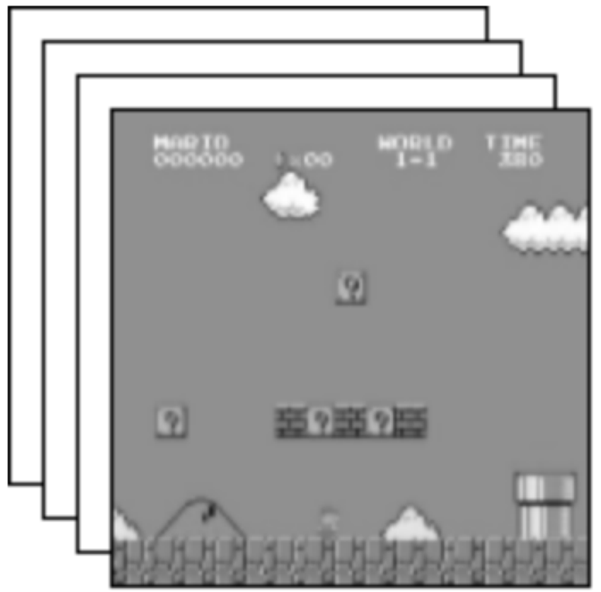
- None 🧍
- Move Right ➡️
- Run Right 🏃➡️
- Jump + Move Right ➡️⬆️
- Jump + Run Right 🏃➡️⬆️
- Jump ⬆️
- Move Left ⬅️

Preprocessing of the State

[3 x 240 x 256]

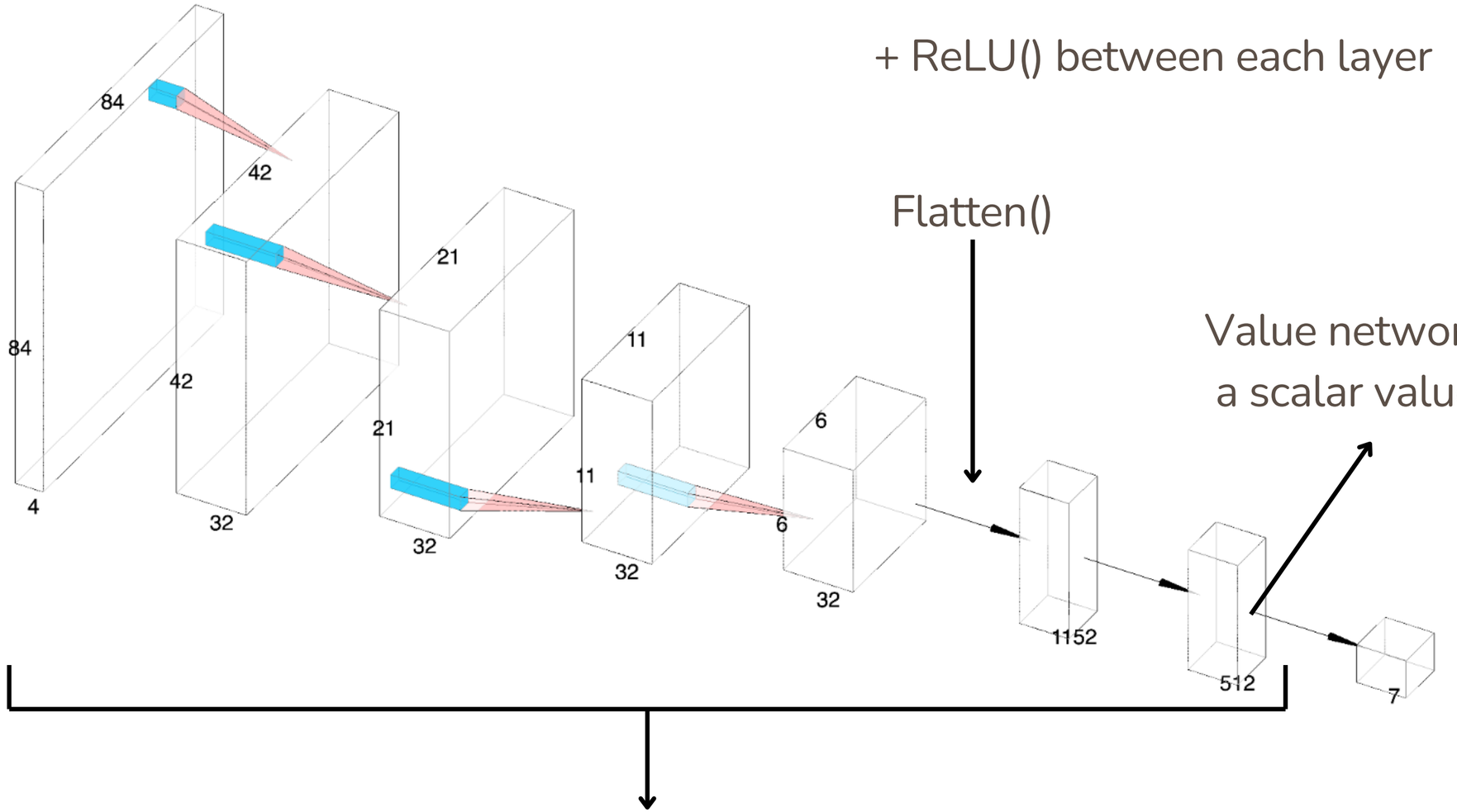
Resize(),
Grayscale(),

FrameStack(4),
Normalise()



[4 x 84 x 84]

Network Architecture



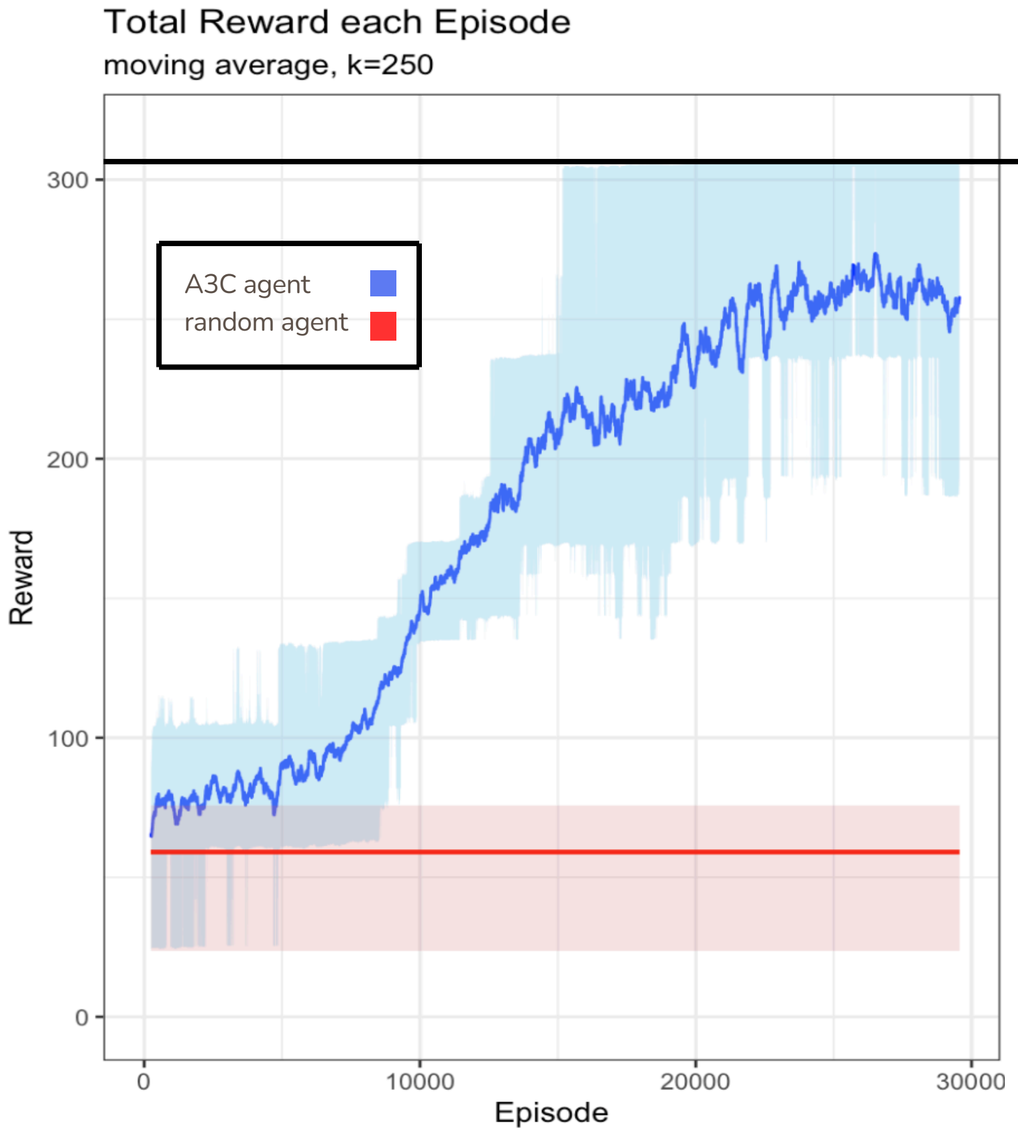
+ ReLU() between each layer

Flatten()

Value network outputs a scalar value

Shared convolutional structure and first dense layer for policy network and value network

Training with dense rewards



Best possible reward (level completion)

Hyperparameters:

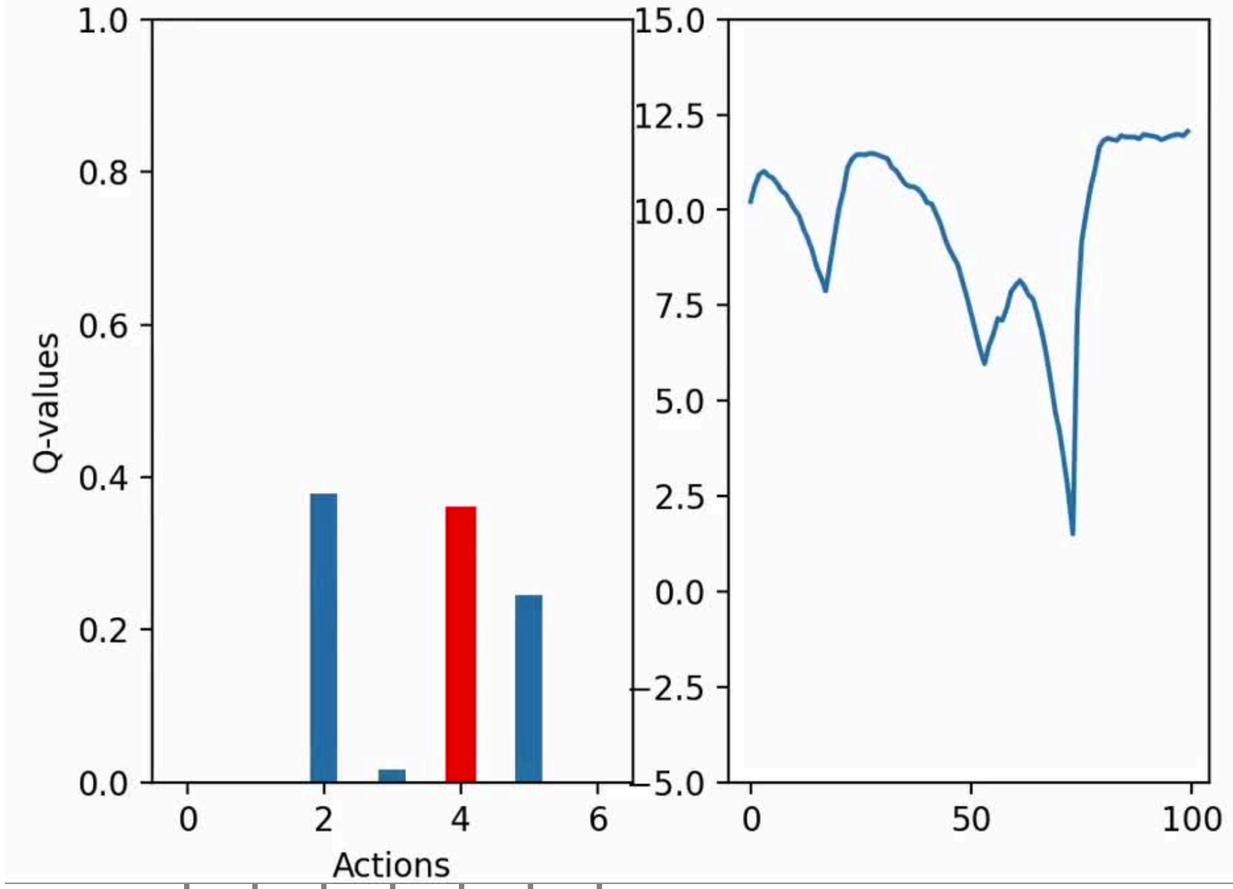
- Learning rate = 1e-4
- Adam Optimiser
- ɣ (discount factor) = 0.9
- Local steps = 50
- β (entropy) = 1e-2

Training Specifics:

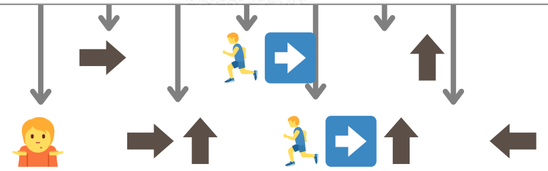
- 30k episodes
- 130k gradient updates
- 24 hours on Orfeo V100

How well does it play?

policy



state-value function



Discussion

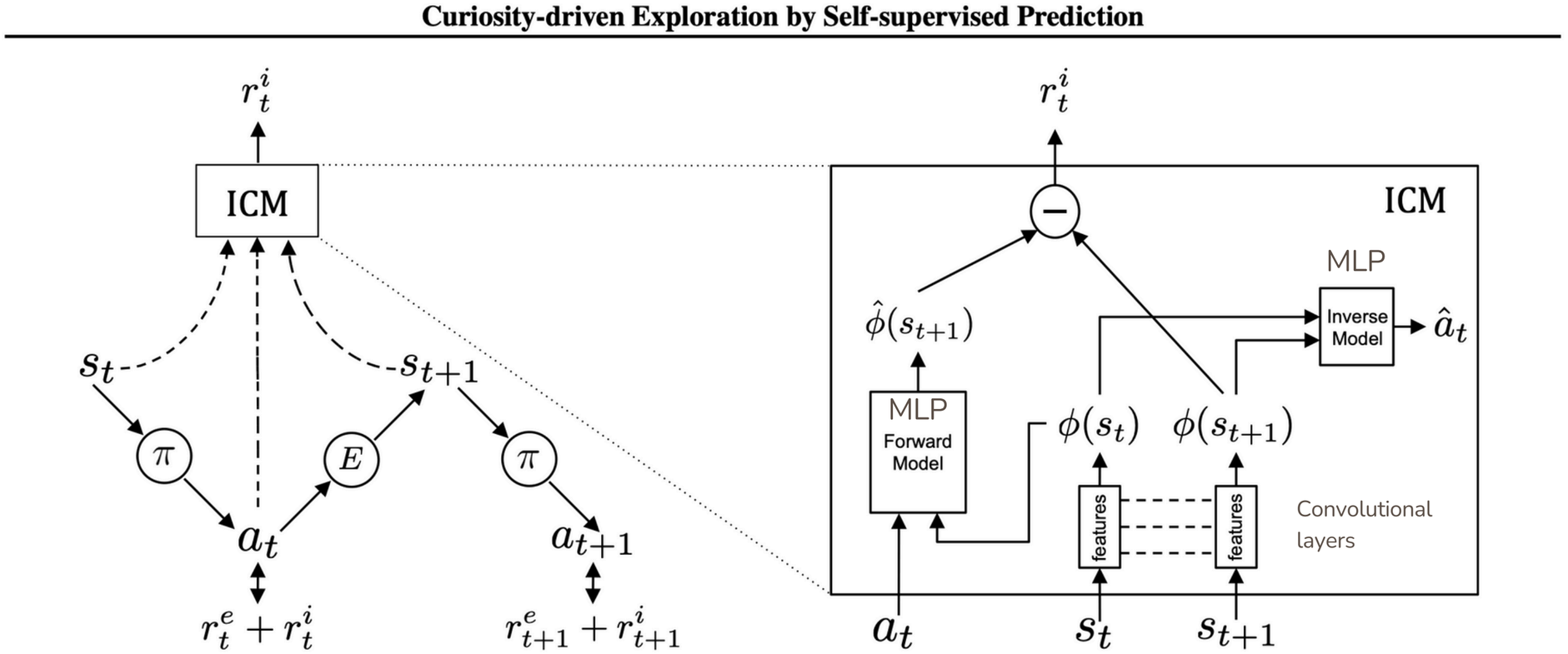
- The agent is able to complete the level fairly consistently with a win rate of 0.71 $\pm$ 0.006 calculated on 600 episodes

- The fact that we obtain a stochastic policy in a deterministic environment might indicate the necessity of more training and it is likely influenced by by the entropy in the training objective and the fact that different actions lead to very similar outcomes / rewards

- The value returned by the network mirrors the sum of the discounted rewards the agent expects to get in the following steps, the weight that future rewards have on the state value function is largely influenced by our choice of gamma

The ICM for environments with sparse rewards

Main Idea: train another model that learns to predict the next state given the old state and the action taken by the agent. Use the error in the prediction as an intrinsic reward for the agent. If the agent has not learnt to predict where it will end up after an action it should probably explore that state-action pair more.



The ICM

Inverse Dynamics Module:

- Takes two subsequent states as inputs and outputs the action that led from the first to the second $\hat{a}_t = g\left(s_t, s_{t+1}; \theta_I\right)$
- Optimisation objective: $\min_{\theta_I} L_I(\hat{a}_t, a_t)$ ⟶ Cross Entropy in our model

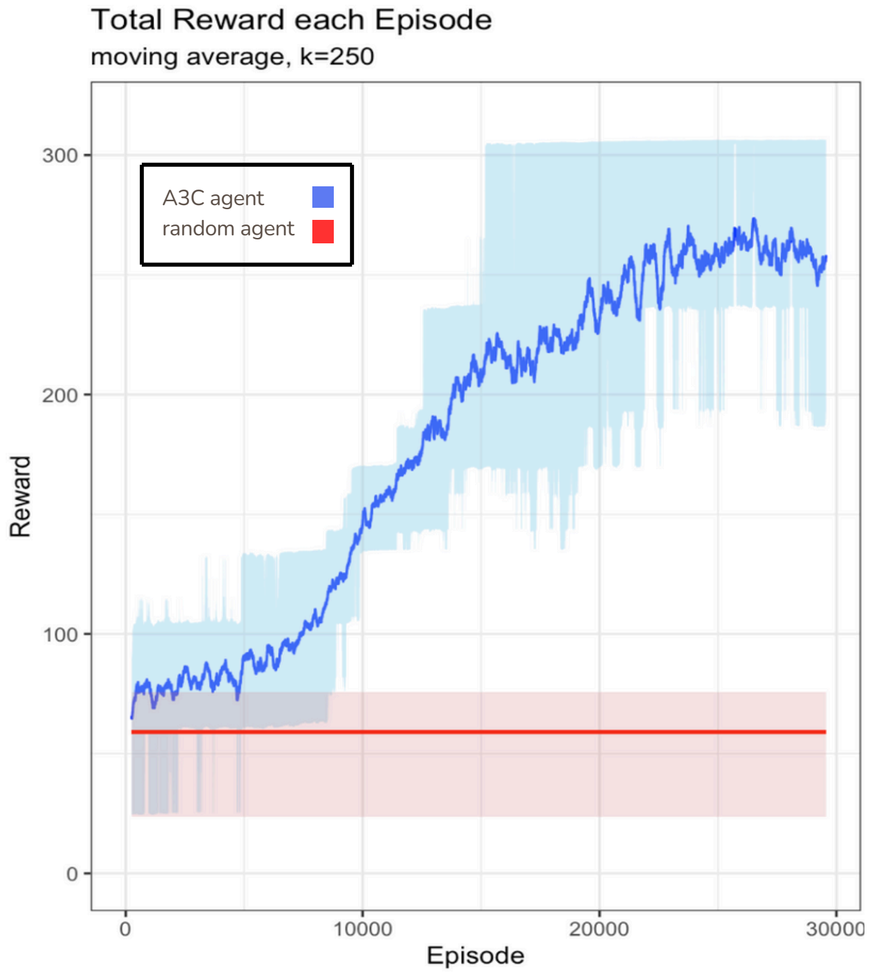
Forward Model:

- Takes the feature representation of the state and an action, and outputs the representation of the state reached $\hat{\phi}(s_{t+1}) = f\left(\phi(s_t), a_t; \theta_F\right)$
- Loss function: $L_F\left(\phi(s_t), \hat{\phi}(s_{t+1})\right) = \frac{1}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$
- Scaled intrinsic reward: $r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$

Final optimisation objective: $\min_{\theta_P, \theta_I, \theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t; \theta_P)}[\Sigma_t r_t] + (1-\beta) L_I + \beta L_F \right]$

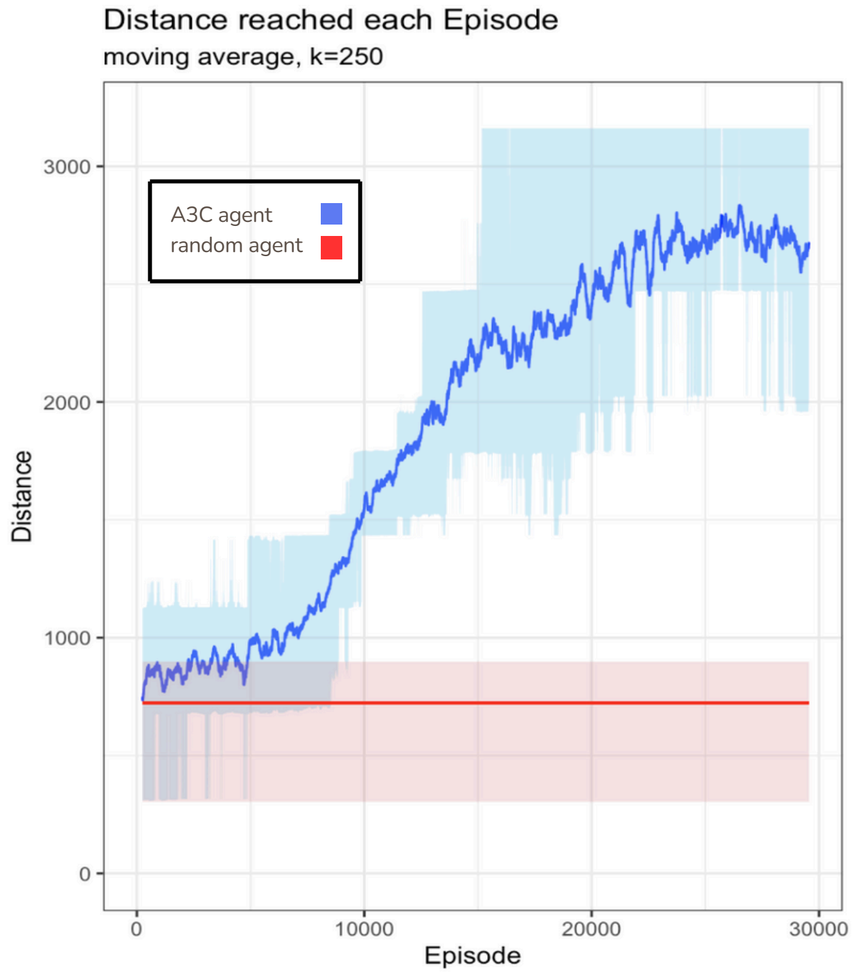
Evaluation without rewards

The rewards from the Gym environment are highly correlated with the distance reached by the agent



From reward

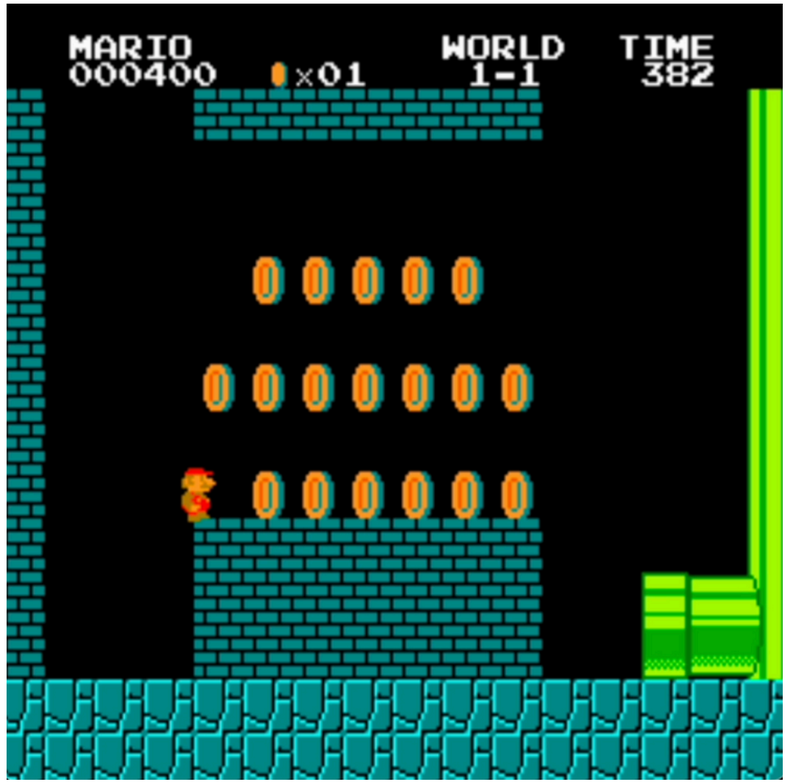
to distance



The pipe mechanism

Entering the pipe at 900

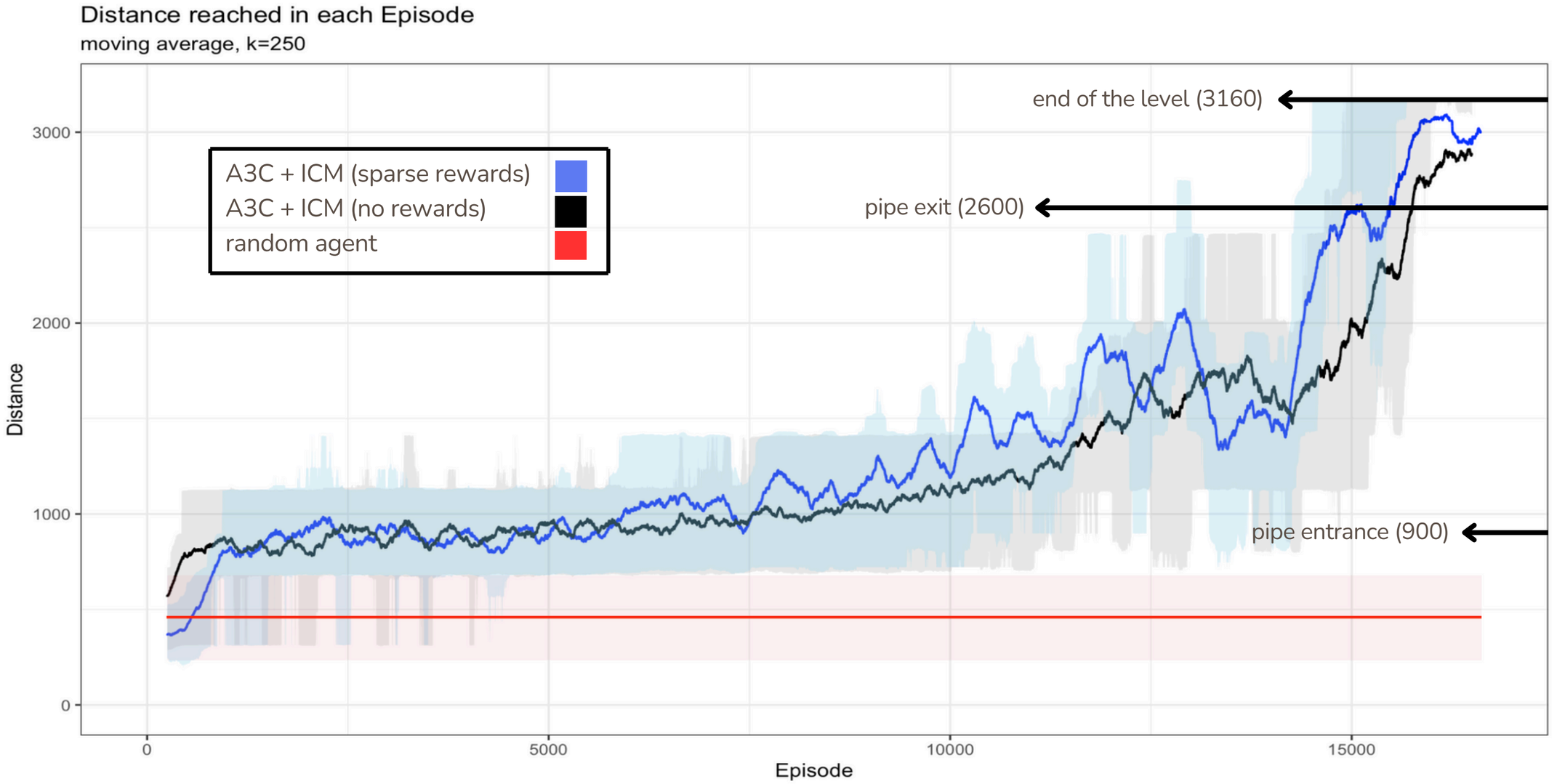
Inside the pipe [0 - 200]



Leaving the pipe at 2600

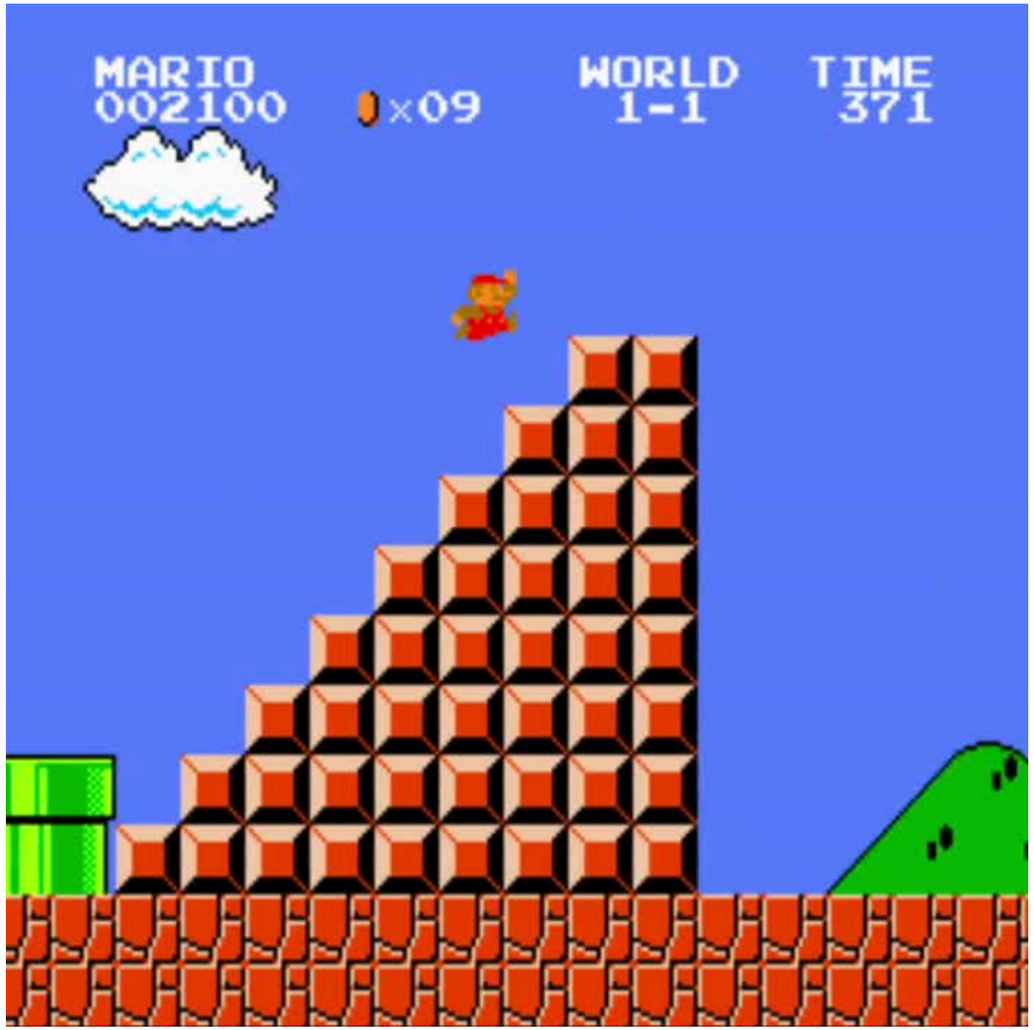
*The agent can only enter the pipe with the extended action set

Training with sparse rewards / no rewards



How do they play?

Sparse Rewards Agent



No Rewards Agent

Further work

- Train an agent with the ICM module in an environment dense of rewards and compare it against the plain A3C agent
- Assess whether the gained knowledge helps the agent explore subsequent levels faster than when starting from scratch.

Failures

- We tried different flavours of DQN
  - Fitted DQN
  - Double DQN
  - Double Duelling DQN

  playing with the following hyper parameters: learning rate, size of the memory buffer, epsilon, epsilon decay, gamma.
  All the agents seem to learn a decent but “fixed” policy, one that does not change with the values of the states

References

- Mnih et al. **“Asynchronous Methods for Deep Reinforcement Learning”**
- Schulman et al. **“Proximal Policy Optimization Algorithms”**
- Pathak et al. **“Curiosity-driven Exploration by Self-supervised Prediction”**
- van Hasselt et al. **“Deep Reinforcement Learning with Double Q-learning”**
- Wang et al. **“Dueling Network Architectures for Deep Reinforcement Learning”**
- PyTorch Super Mario webpage