IDSIA
Draft March 28, 2019

# Curiosity: Learning from the Program of the Forward-Dynamics Model
## (Technical Report)

R. Malm

*Swiss AI Lab IDSIA*
*Galleria 2, Via Cantonale 2c, CH-6928 Manno*

**Abstract**

In contrast to the *classical* curiosity-driven reinforcement-learning (RL) agent, we investigate the idea that the agent has access to the program of the forward-dynamics model of the environment which might be beneficial in finding an improved exploration strategy. This idea has been proposed by J. Schmidhuber. All experiments have been conducted by the author of this report. The report presents the results of working for 2.5 months at IDSIA on the topic *curiosity* in deep reinforcement learning.

# 1 Introduction

An RL agent typically needs to explore its environment in some way to gain experience that allows it to learn better and eventually optimal policies. One distinguishes *undirected* exploration methods, which rely on random actions (such as $\epsilon$-greedy), from *directed* exploration strategies, that are mostly based on the principle of optimism in the face of uncertainty. Here, we focus on the latter form of strategies in the form of intrinsic motivation/rewards for the agent, which become critical whenever extrinsic rewards of the environment are sparse. Most formulations of intrinsic rewards can be grouped into two broad classes that 1) encourage the agent to explore 'novel' states, and 2) encourage the agent to perform actions that reduce the error/uncertainty in the agent's ability to predict the consequences of its own actions. The former class is based on the idea of *visitation counts* [7–10] which discourage the agent from revisiting the same states, while the latter one is based on using the prediction error of a forward-dynamics model of the environment as the reward signal [1, 2, 4–6].

In this report we focus on the second class, the *curiosity-driven* RL agent, which requires building a model of the environmental dynamics that predicts the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ executed at time $t$. [2] is a recent paper along these lines, which includes a forward and inverse dynamics model as well as feature representations for the states, and investigates curiosity in sparse extrinsic reward environments like *Super Mario Bros.* and *Viz-Doom*. For the sake of clarity in this report we will neglect the inverse dynamics model and the explicit feature representations of the states, which is justified due to the simplicity of the considered environment in the experimental section.

In this case, termed the *classical* approach to curiosity, the agent receives an intrinsic reward $r_t^i$ at time $t$, which is proportional to the model's prediction error $\frac{1}{2}||s_t - \hat{s}_t||_2^2$, where $\hat{s}_t \equiv \hat{s}_t(s_{t-1}, a_{t-1}; \theta_M)$ is the prediction of the forward model given the previous state $s_{t-1}$ and action $a_{t-1}$, as well as the model parameters $\theta_M$. The

intrinsic reward $r_t^i \equiv r_t^i(s_t, \hat{s}_t)$ is then additively combined with the extrinsic reward $r_t^e(s_t)$ of the environment, such that the total reward is given by $r_t = r_t^i + r_t^e$. The agent's probabilistic policy $p(.|s_t; \theta_P)$, parameterised by $\theta_P$, is then updated such that the agent's action probability for a given $(s_t, a_t)$ pair is increased or decreased according to the implemented RL algorithm. In the following we will refer to the forward dynamics model of the environment as the *predictor*.

# 2 Idea

The core idea is that the curiosity-driven RL agent is allowed to see the program of the predictor in order to *surprise/fool* the predictor, i.e. by taking an action where the model's prediction error is large in order to receive a high intrinsic reward. However, since the agent does not know the true next state the agent cannot directly learn to predict the prediction error. Given the current state and the current predictor's program, the agent needs to learn which action will lead to the predictor's smallest confidence in its own next state prediction. Taking this action, where the predictor is most uncertain about its own prediction, will probably result in a large prediction error and therefore high intrinsic reward.

In other words, the agent should use the predictor's program to *implicitly* learn a (model-based) planning strategy to surprise/fool the environmental model. The word *implicitly* is highlighted here, since it means that the agent (or *self-model*) is not trained to predict the future prediction error of the predictor (or *world-model*), see [11] for a recent paper that *explicitly* trains the agent to predict the world-model's loss.

Now, in practical terms our idea implies that we feed the current weights $\theta_M$ of the predictor at time step $t$ as an additional input, besides the environmental state $s_t$, to the agent, such that the agent's action probability is given by $p(a_t|s_t, \theta_M; \theta_P)$ for an action $a_t \in \mathcal{A}$. Theoretically, the agent *might*[1] have the ability to infer the uncertainty of the predictor in the next state $s_{t+1}$ given the triple $(s_t, a_t, \theta_M)$. The corresponding algorithm is shown below.

**Algorithm 1:**

initialise $\theta_M$, $\theta_P$
**for** *each episode* $k = 1, 2, ...$ **do**
  reset environment; reset agent buffer; fix $s_0$ to start state;
  **for** *each time step* $t = 0, 1, 2, ..., T-1$ **do**
    sample action $a_t$ using agent policy $p(.|s_t, \theta_M; \theta_P)$
    take action $a_t$ in environment and observe next state $s_{t+1}$, reward $r_{t+1}^e$, done flag $d_t$
    use predictor to compute intrinsic reward $r_{t+1}^i$, and then total reward $r_{t+1}$
    add tuple $(s_t, \theta_M, a_t, s_{t+1}, r_{t+1}, d_t)$ to agent buffer
    add tuple $(s_t, a_t, s_{t+1})$ to predictor buffer
    update predictor with transitions $(s_i, a_i, s_{i,next})$, $i = 1, 2.., N$ uniform rand. sampled from the predictor buffer:

$$\theta_M \leftarrow \theta_M - \alpha_M \frac{1}{N} \nabla_{\theta_M} \sum_{i=1}^{N} \frac{1}{2} ||s_{i,next} - \hat{s}(s_i, a_i; \theta_M)||_2^2$$

  **end**
  update weights $\theta_P$ of the agent policy via PPO using the current trajectory from the agent buffer
**end**

It is worth emphasising, that the agent does *not* need and should *not* learn to execute the predictor's program

[1] We have weakened this statement, since there is a caveat in the argument because the predictor is not completely defined solely by its weights.

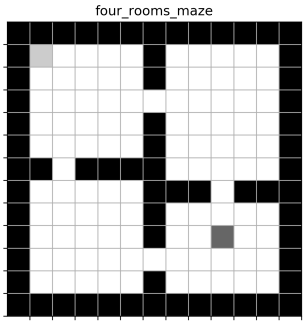


Figure 1: Four-rooms maze environment. The start position of the agent is shaded in light gray, while the goal position is shaded in dark gray. The walls are shaded in black.

internally, since relevant is only the uncertainty in the predicted next state and not its actual value. Thus, the agent has to learn its own metric of measuring uncertainty. This might be helpful in addressing the general inherent limitation of prediction error based curiosity in case of stochastic environments. According to the classical curiosity-driven approach, if the transitions in the environment are random, then even with a perfect dynamics model, the expected reward will be the entropy of the transition, and the agent will seek out transitions with the highest entropy. This is also known as the noisy-TV problem [4]. But, if the agent can see the predictor's program it might realise a certain repeating pattern and take actions to get unstuck from stochastic environmental dynamics. However, the noisy-TV problem is beyond the scope of this report.

Finally, it is worth mentioning that we can combine the state space $\mathcal{S}$ of the environment with the predictor program space $\mathcal{P}_{\mathcal{M}}$ defining a new observation space $\mathcal{O} = \mathcal{S} \times \mathcal{P}_{\mathcal{M}}$. In this case the total reward at time step $t$ is only dependent on the observation $o_{t-1} \in \mathcal{O}$ and action $a_{t-1} \in \mathcal{A}$ at the previous time step $t-1$. Thus, the whole problem can be modelled as a Markov-Decision Process (MDP). However, this statement is not strictly true anymore if we replace the predictor's program space $\mathcal{P}_{\mathcal{M}}$ by the predictor's weight space $\Theta_M$. The reason is that knowing the weights of a network is not sufficient to fully describe the network, since information about the topology, e.g. weight connections or activation functions, is missing.

# 3 Experiments

We implement deep RL agents, where the probabilistic policy is a feed-forward neural network (2 hidden layers of each 32 units) which is trained via PPO [3]. All PPO agents are implemented using the generalised advantage estimate [12] with $\gamma = 0.99$ and $\lambda = 0.97$ as a baseline, which can be calculated from the agent's rewards and the state-value function. The clipping range is fixed to 0.2 for the PPO surrogate loss function. The predictor is a feed-forward neural network (1 hidden layer, 2 hidden units as a default) trained via supervised learning with a mean-squared error loss. Intrinsic rewards are clipped to the range $[0, 0.1]$. Each neural network is optimised with Adam where we set the maximal learning rate to 0.0005 for the agent networks and 0.001 for both the state-value function and predictor networks.

As the environment we choose the four-rooms maze, that has 104 distinct states, which the agent is allowed to occupy. The *start* state is always localised in the upper left corner while the *goal* state is fixed within the lower right room, see Figure 1. Each trajectory $\tau = s_0, a_0, s_1 ..., s_{T-1}, a_{T-1}, s_T$ has a fixed and finite time horizon $T = 21$, which is the minimal number of steps required for the agent to visit any state within one episode. If the agent visits the goal state it receives an extrinsic reward of 10 once, otherwise zero extrinsic reward. The shortest path to the goal state comprises 17 time steps.

We consider three main scenarios. In the first scenario the agent has no access to the predictor's weights

and is trained via PPO using the intrinsic rewards. This corresponds to the *classical* curiosity approach, and is referred to as the *ppo_predictor* scenario. The second scenario differs from the first one by additionally feeding randomly fixed weights of the same dimension as the weights of the predictor into the agent. This case is referred to as the *ppo_predictor_fixed_weights* scenario. Finally, the third scenario is the agent, that receives the current weights of the predictor at each time step. This corresponds to the proposed idea in this report, and is referred to as the *ppo_predictor_weights* scenario.

The upper plot in Figure 2 shows the extrinsic return for each episode during training for *rs* (random search), *ppo* (just the PPO agent without intrinsic rewards), *ppo_predictor_fixed_weights* and *ppo_predictor_weights*. In generating the plots we have used optimal hyper-parameters and performed 5 runs with different seeds for each of the agents. Clearly, *rs* and *ppo* cannot efficiently explore and solve the maze problem. The agents of the three scenarios have comparable performance, slowly increasing the extrinsic return with each episode. The curves are fluctuating, which can be explained by the encoding of each environmental state as a $(x, y)$ tuple where each coordinate is normalised to the range $[-1, 1]$. Thus, the agent has to precisely learn small changes in $(x, y)$ in order to learn how to reach the goal state. The lower plot shows the number of distinct states visited by the agent in the past and can be interpreted as a measure for exploration (in analogy to visitation counts) independently of extrinsic rewards. Again, we see that while *rs* and *ppo* perform poorly, the three main scenarios perform similarly.

Figure 3 differs from Figure 2 by encoding the states as one-hot vectors. Interestingly, the agent of the *ppo_predictor* scenario can quickly learn a robust policy to reach the goal state. On the other hand, the additional input of the predictor's weights prevent the remaining agents to efficiently learn a robust policy that achieves the goal state. The signal from the sparse representation of the states is very weak compared to the dense representation of the predictor's weights.

Figure 4 shows that the number of hidden units, which is related to the capacity, of the predictor has a strong effect on the performance of the *ppo_predictor_weights* scenario. Clearly, if the capacity is too small (no hidden units) the environmental dynamics model can not be learned sufficiently by the predictor and the guidance through intrinsic rewards leads to a suboptimal exploration strategy for the agent. On the other hand, if the capacity is too large the additional weight input to the agent diminishes its capability to learn a robust policy that efficiently reaches the goal state. The best performance for the agent of the *ppo_predictor_weights* scenario can be achieved if the predictor network has two hidden units.

# 4 Conclusion

None of the experiments has shown any hint that the agent, which can see the predictor's weights, is able to learn an improved exploration strategy allowing it to solve the four-rooms maze problem more efficiently. On the contrary, the additional weight input even diminishes the performance of the agents if the environmental states are one-hot encoded or if the number of predictor weights gets too large.

Conceptually, it should be mentioned that the predictor's weights are not sufficient to fully describe the predictor, since the topology of the predictor's network (weight connections, activation functions, etc.) is not captured.

While the idea, that an agent might be able to use the predictor's program in order to implicitly learn a planning strategy for achieving higher intrinsic rewards, sounds intuitive, finding a working implementation of this idea probably requires a more principled formulation as well as an extensive amount of experimental work.

# References


[1] Y. Burda, H. Edwards, D. Pathak, A. J. Storkey, T. Darrell, A. A. Efros, Large-Scale Study of Curiosity-Driven Learning, CoRR, [arXiv: 1808.04355]

[2] D. Pathak and P. Agrawal and A. A. Efros and T. Darrell, Curiosity-driven Exploration by Self-supervised Prediction, CoRR, [arXiv:1705.05363]

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, Proximal Policy Optimization Algorithms, CoRR, [arXiv:1707.06347]

[4] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In From animals to animats: Proceedings oft he first international conference on simulation of adaptive behavior, 1991.

[5] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In NIPS, 2016.

[6] S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In NIPS, 2015.

[7] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In NIPS, 2016.

[8] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In NIPS, 2012.

[9] G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. [arXiv:1703.01310], 2017.

[10] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete bayesian reinforcement learning. In ICML, 2006.

[11] Learning to Play with Intrinsically-Motivated Self-Aware Agents A Haber, Nick; Mrowca, Damian; Fei-Fei, Li; Yamins, Daniel L. K. arXiv:1802.07442

[12] T High-Dimensional Continuous Control Using Generalized Advantage Estimation A Schulman, John; Moritz, Philipp; Levine, Sergey; Jordan, Michael;Abbeel, Pieter arXiv:1506.02438

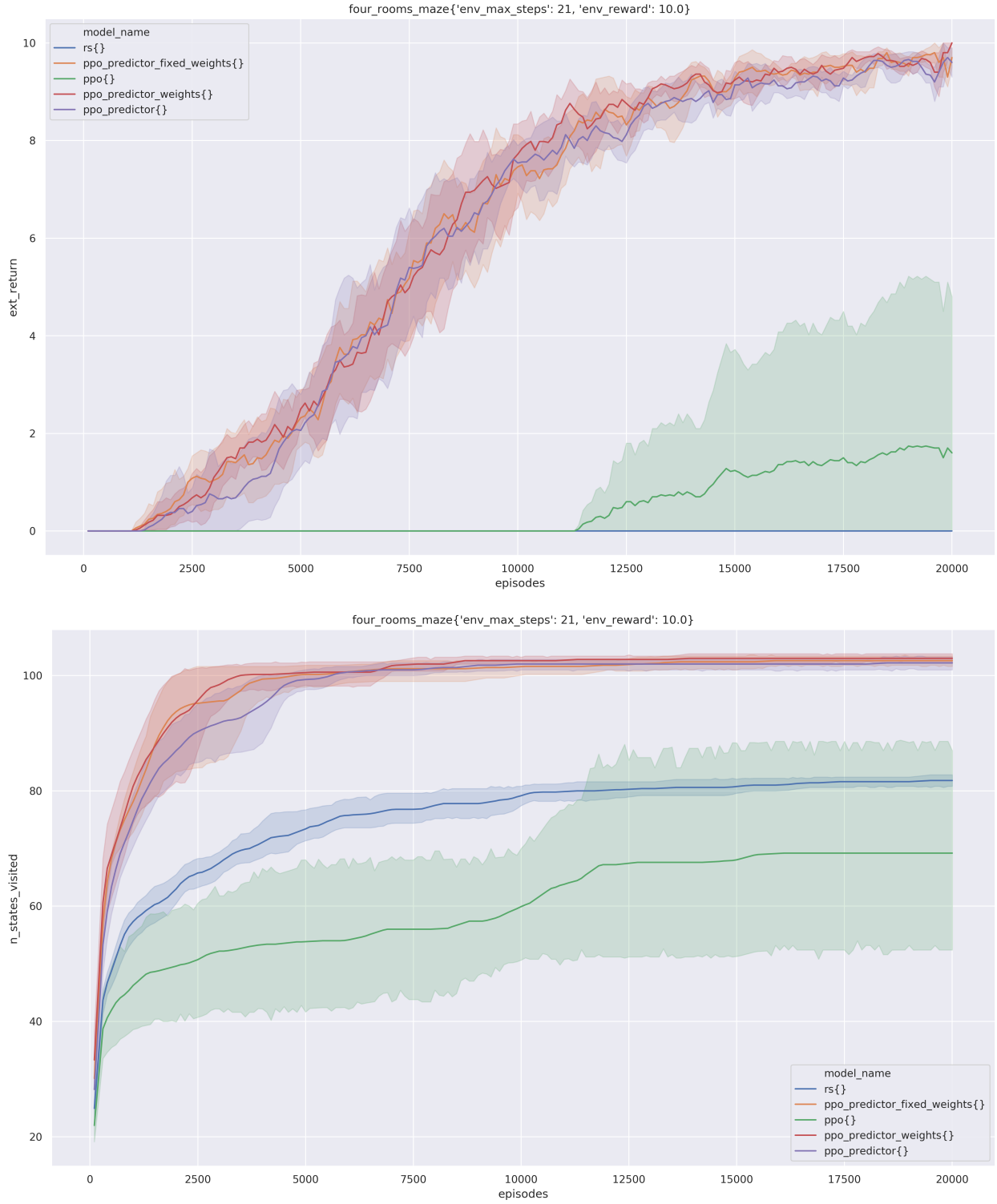


Figure 2: The upper (lower) plot shows the extrinsic return (number of distinct states visited by the agent) versus the number of episodes during training. The predictor network has one hidden layer with 2 units for all agents. The states are encoded as $(x, y)$ coordinates. See the text for further details.

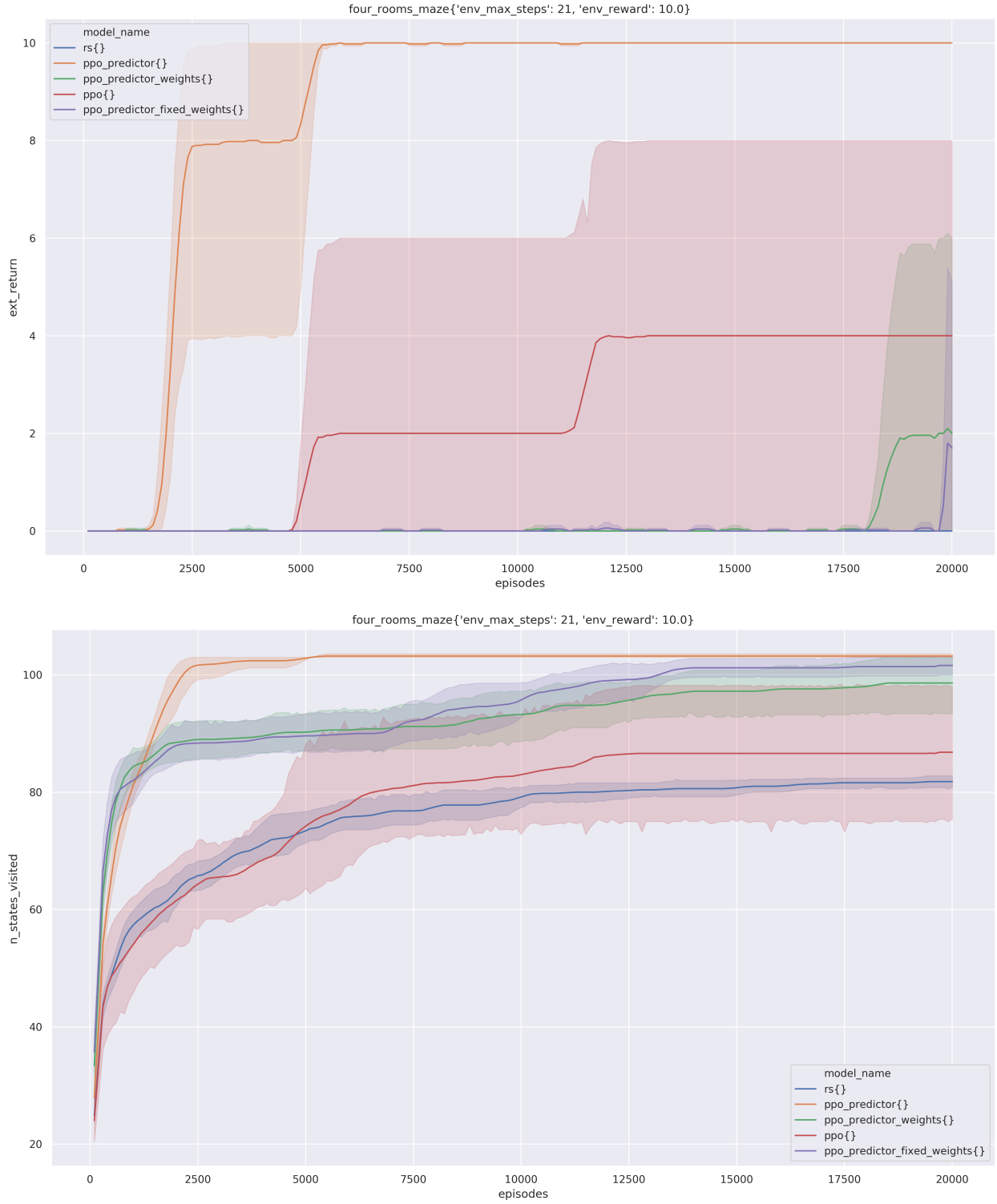


Figure 3: In contrast to Figure 2, the states are encoded as one-hot vectors. The predictor network has one hidden layer with 32 units for the *ppo_predictor* scenario, otherwise 2 hidden units. The agent of the *ppo_predictor* scenario can quickly learn a robust policy, while the remaining agents perform poorly. See text for further details.


Figure 4: The *ppo_predictor_weights* scenario is shown with varying number of hidden units ([]: no hidden units,: [2]: 2 hidden units, ...) for the predictor network. The states are encoded as $(x, y)$ coordinates. Clearly, the scenario with 2 hidden units for the predictor network performs best in the upper plot. See text for further details.