State description language and PathRL

DMITRII BABAEV, AIRI Sber AI Lab , Moscow, Russia

This paper is a work-in-progress description of the PathRL algorithm, an application of the hierarchical reinforcement learning to the arbitrary complex environments with sparse rewards. It includes high-level planning and low-level plan execution.

1 PathRL algorithm

1.1 Training scheme

This goal-setter receives only the current state and produces the set of predicates $e_g = G(s_t)$ that lead to potentially high rewards, i.e. G learns to generate the set of predicates that describe high-reward states. During this phase actions are produced by pre-trained navigator $a_t = N(s_t, G(s_t))$ with frozen parameters, yet differentiable w.r.t. the goal-state embedding. This forces the goal agent G to reproduce and communicate in the same description "language" as the encoder agent. This goal-setter is also penalized for the non-reachable goals which worker is failed to achieve.

1.1.1 Reward scheme. The goal-setter G is trained with four rewards: the extrinsic reward, i.e. true reward of the environment (3); the intrinsic curiosity reward for active exploration (4); non-reachable goal penalty, received if the worker is not achieved the generated goal in k steps; and short goals penalty, for goals, achieved in less than c steps (2).

$$\text{in}(s, g) = \text{sim}(E(s),\ g) \geq \sigma \tag{1}$$

$$\mathcal{R}_{\text{gs}}^{(g_i)} = \sum_{t=0..k,\ g=g_i} \begin{cases} 1 & t > c \wedge \text{in}(s_t, g) \\ -\epsilon_s & t \leq c \wedge \text{in}(s_t, g) \\ -1 & t = k \wedge \text{in}(s_t, g) \\ 0 & \neg\text{in}(s_t, g) \end{cases} \tag{2}$$

$$\mathcal{R}_{\text{ext}} = \sum_{t \geq 0} \gamma^t r_{t+1}^E \tag{3}$$

$$\mathcal{R}_{\text{int}} = \sum_{t \geq 0} \text{surp}(s_{t+1}) \tag{4}$$

In this scheme G is encouraged to propose queries for states of two kinds: *explotable* states with high extrinsic reward, and *explorable*, unknown "terrritory" states high curiosity potential. The combination of curiosity reward scheme and navigator agent have some parallels with Go-explore algorithm [8], where the navigation is done via virtual environment resets.

1.1.2 Navigator training. The navigator can be pre-trained during preliminary phase, and then, additionally trained during the main phase. The navigator can also be trained from scratch during the main phase, without pre-train during the preliminary phase.

During the navigator training, the navigator weights are unfrozen. The navigator is trained on the intrinsic rewards of reaching the generated goals. The navigator does not receive reward for too complex goals, when goal is not achieved in k steps (5).

E-mail: dmitri.babaev@gmail.com.
Author's Contact Information: Dmitrii Babaev, AIRI and Sber AI Lab , Moscow, Russia.

$$\mathcal{R}_{\text{nav}}^{(g_i)} = \sum_{t=0..k,\ g=g_i} \begin{cases} 1 & \text{in}(s_t, g) \\ -\epsilon & \neg\text{in}(s_t, g) \end{cases} \tag{5}$$

It is also possible to use a form of navigator pre-train similarly to HER [1]. In this setup, the navigator occasionally receives "return to the state" tasks from the state history buffer.

1.2 Navigation reward

One possible training scheme is to shape the reward so that it encourages moving closer from the current state s_t to the target state s_g. The notion of state *proximity* must take into account the fact that states reside on certain latent manifold, must be consistent with respect to environment's randomness, and at the same time focus on controllable and abstract away uncontrollable or irrelevant features of the states. For example, positively rewarding for diminishing Euclidean distance between s_g and s_t fails all these desirable properties.

Specifically, in Random Disco Maze navigation environment, [2], the pixel-wise distance would hinder the task completion due to irrelevant random color of the labyrinth's walls. In the same environment the initial and goal states that are separated by an adjacent wall, yet unreachable from one another would be considered close states. Finally, in Pac-Man[1] the position of the player within the maze is a controllable feature, while the positions of patrolling ghosts are not, yet the distance between states would be sizeable, even if the protagonist has arrived exactly at the specified target position. Similarly with Freeway[2], the player's car is controllable, while the opponents' cars are not, [6, fig. 1].

One possibility is to define a proximity score as $d(s_a, s_b) := \frac{\phi(s_a)^\top \phi(s_b)}{\|\phi(s_a)\| \|\phi(s_b)\|}$ – the cosine similarity between the latent representations of states. The mapping $\phi \colon \mathcal{S} \to \mathbb{R}^{d_h}$ is a learnt state embedding that removes features irrelevant to the navigator's task and effectively flattens the state manifold. The embedding ϕ, that drops uncontrollable features can be trained in self-supervised manner on the inverse dynamics task. i.e. inferring the action taken between consecutive states, [2, 6].

Another scheme is to split a target state to the overlapping parts in sliding-window manner. Then, it is possible to reward agent if some part of the current state became same as the corresponding part of the target state and penalizie agent if some part was the same and became different. The advantage of this scheme is that the observation encoder is not needed. The main challenge of this scheme is the requirement to generate the whole observation as a goal for a goal-setter. The possible way to simplify this task is to pre-train goal-setter on the pairs of observation from the same episode. I. e. pre-train it on the task to generate the representation of the step t observation taking the representation of the step t-i observation.

1.3 World state representations

The most promising possibility, especially in the context of the desiderata outlined in sec. 1.2, is to use compressed abstracted representations of the environment states in the encoder, the navigator and the goal-setter agents. The state embeddings can be produced by a world model pre-trained by the next state prediction task, [10], by an inverse dynamics model, [2], or by a contrastive model as in [25].

[1] https://en.wikipedia.org/wiki/Pac-Man_(Atari_2600)
[2] https://en.wikipedia.org/wiki/Freeway_(video_game)

1.4 Motivation

The proposed scheme should be effective in sparse extrinsic reward settings: the low-level basic interaction in the environment, when trained to solve the environment on its own, would experience the detrimental effects of reward sparsity most acutely. However, if instead we train a low level agent on state-navigation tasks, motivating it for achieving the goals (a sort of synthetic intrinsic reward, but not curiosity-based), and delegate the solution of the environment itself to an abstract planning agent, which collects extrinsic rewards, we could effectively significantly reduce the planer's reward sparsity. If the high-level agent uses relatively small set of high-level actions (goals) then the number of possible trajectories to explore is drastically reduced: the trajectories of 10 steps of high-level actions will be used instead of trajectories of 100 steps of the low-level actions.

By using manager-worker scheme we are able to learn efficient reliable navigation skills using synthetic navigation tasks, without any extrinsic reward. The proposed scheme should be effective for the multi-task learning setting in the same environment: a well trained and capable navigator model, even though trained for one specific environment and goal state distribution, can be effectively reused across different tasks.

As shown by Ecoffet et al. [8] the exploration of distant states greatly improves the algorithm effectiveness. The availability of navigator agent enables training a goal agent, which is encouraged to explore remote states. In contrast to [8], though, by using learnt state representations that abstract away irrelevant information, and by decoupling goal-setting from goal-achieving the scheme proposed in this work may be applied to stochastic and multi-task environments as well.

1.5 Interpretable RL

The proposed PathRL algorithm can be made interpretable by adding state representation decoder, that would transform a goal back to the desired state, associated with that goal. This can be done training state decoder together with state encoder during the first phase of training.

1.5.1 Discriminator loss for goal-setter. To force goal-setter to generate goals, that are opservation representations, additional losses can be used. One option is to use discriminator loss. Separate discriminator net can be trained to distinguish between the generated goals and state representations and its accuracy can be used as loss to goal-setter similarly to the GAN training scheme (6). Another option is to pre-train goal-setter on the pairs of observation from the same episode. I. e. pre-train it on the task to generate the representation of the step t observation taking the representation of the step t-i observation.

$$min_{\Theta_M} max_{\Theta_D} L(\Theta_M, \Theta_D) = E_{s \in P_s} \log D_{\Theta_D}(E(s)) + E_g \log(1 - D_{\Theta_D}(M_{\Theta_M}(s))) \tag{6}$$

1.5.2 Why goal-setter goals can provide interpretation. Ideally trained goal-setter should output goals that are keypoint states. The keypoint state of the pair of states s_b and s_e, is the state, that can should be included to any trajectory between s_b and s_e. The keypoint state of some environment is the state with high *betweenness* measure, i. e. the state that should be included in many possible trajectories for this environment.

2 Semantic observation attributes

The alternative, and possibly more scaleable and interpretable approach to PathRL is to use the set of facts about the observation instead of its representation. Each fact is defined as a semantic triple (subject, predicate and object), where first and third terms are taken form the same fixed-size set of possible tokens A and second term is taken from another fixed-size set of possible tokens B.

On the first phase the attribute *difference* between the current observation and some random observation is calculated. A random subset of attributes from that difference is selected and passed to the worker. The worker goal is to reach any state which have the passed set of attributes. On the second phase the task of goal generator is to produce the subset of attributes that would maximize the reward.

Semantic observation attributes can be manually engineered, or possibly mined from the observations. Some environments. e. g. UI task environments, like MiniWob++ [16] or computer game environments, like NetHack [14] already have highly structured states, that can be used as the replacement of the fact sets.

If the observations are 2D images, it is reasonable to split them using some grid and define an unidependent set of semantic attributes for each cell of the grid. This would allow to preserve the spatial information from the original observation.

2.1 Image guessing

Observation attributes can possibly be mined using approach considered in language emergence papers. In this scenario the image attributes is learned in the unsupervised setting of image guessing game similarly to Havrylov and Titov [12]. In Havrylov and Titov [12] the image is transformed into the sequence of tokens, in PathRL, the image is transfromed to the set of facts.

2.2 Discrete world model

The possible approach to extract state entities is to train the global state encoder to produce a fixed set of the discrete vectors or global vectors. The global vectors will be used for the restoration of the entire sequience of states along some trajectory of the state-action pairs. The global vectors are produced by encoding the initial state of the trajectory. The VQ-VAE like approach can be used to make global vectors discrete.

The state restoration model is the recurrent network, that, on each step, takes the global vectors and the current action and outputs the next state prediction. The initial hidden vector of the RNN is produced from the initial state by the dedicated hidden vector encoder. The state difference loss is used to train the global state encoder, the state restoration RNN, and the initial hidden vector encoder in the end2end manner. The system does not take the environment states as its input, except the initial state.

2.2.1 Discrete space autoencoder. Another option is to introduce a sort of a spatial prior and, instead of predicting the next state, predict the current state in the autoencoding manner. In this case, the current state encoder outputs the matrix of the categorical distributions over the global vectors. In this case, each cell of the output contains an attention vector, that will be applied to calculate the weighted sum the global vectors. A form of the dot-product attention can be used: the dot products of the output vector of the current cell and each of the global vectors, normalized via the softmax will be used as the weights of the global vectors. Finally, matrix of the weigthed sums of the global vectors is used to produce the current state prediction. Here, the trajectory is used as the source of a different combinations of the same entities, the actions are not used at all.

2.3 Search in semantic observation space

The perespective approach is to use neural net guided MCTS to find the valuable combination of semantic attributes. The desired change in any single observation attribute can be used as a high-level action, and will form a branch of the semantic observation tree. In this setup, the goal-setter iteratively proposes changes in the attributes, and builds a search tree to find a high

reward combination of semantic attributes. The navigator task is to reach the proposed high-level state, i. e. implement the proposed change in the attributes.

The desired change in a single observation attribute may also lead to inevitable changes in other attributes, but for the tree search algorithm it is practical to define an action by the single changed attribute.

As in AlphaZero [24] the additional value estimator for the proposed changes in the attributes can be used to guide the search. It takes the updated set of the semantic attributes, associated with the new branch of the semantic tree and returns its accessed value.

The overall algorithm respembles MuZero [22], but, instead of state space, the search is made in the semantic attributes space, and attribute changes are used instead of basic actions. Also, in the basic version, the algorithm interacts with the actual environment instead of the learned model.

The MCTS search can be guided by the following signals: negative penalty if the navigator can not reach the proposed state in k steps, the reward for reaching the desired state, e. g. achieving the assigned task, or winnning the game, the negative reward for reaching the undesired state, e. g. losing the game. In some environments there can be only desired state, on others, only undesired states exist, still, only one kind of the states can be used to guide the search.

3 Related work

Feudal RL by Dayan and Hinton [7]. The task is to find a way in the maze. A hierarchy of policies is defined. A policy use blocks of space as atomic movement positions. Each level of hierarchy use exponentially larger blocks comparing to the previous level. A policy selects a desired block to move to as its action and then assigns a task to lower-level policy. Higher-level policy defines reward for the lower-level policy. The reward for lower-level policy is gained if it successfully guides the agent to the block, defined by the higher-level policy.

STRAW by Vezhnevets et al. [26]. The network holds two structures in memory: action plan A and commitment plan c. T-th column of A contains probabilities to perform specific action at time t. c is a vector of probabilities to update the current plan at time t. If $g \sim c_{t-1} = 1$ then the network performs plan update. Both A and c are refreshed during update. The network is trained using A3C.

Hierarchical Deep Reinforcement Learning by Kulkarni et al. [13]. The meta-controller learns an approximation of the optimal goal policy $\pi(g|s)$, i. e. it receives state s and select a goal g from the set of all possible current goals G. The meta-controller operates on the slower time-scale than the controller. The meta-controller receives extrinsic reward from the environment.

The controller learns an approximation of the optimal action policy $\pi(a|g, s)$, i. e. it receives state s and goal g and selects an action a. The internal critic provides intrinsic reward for the controller. For example, a critic for a game can check that specific conditions are fulfilled, e. g. an agent reaches the door.

The critic and the set of possible goals G are not learned but are considered available.

HIRO by Nachum et al. [18]. Ant Gather / Ant Maze / Ant Push / Ant Fall tasks are used. The higher-level policy instructs the lower-level policy via high-level actions, or goals, which it samples anew every c steps. A higher-level policy takes current state and outputs the goal for lower-level policy. The lower-level policy interacts directly with the environment. The goal simply instructs the lower-level policy to reach the specific state. The reward for lower-level policy is provided if the environment yields an observation close to matching the desired goal.

DADS by Sharma et al. [23]. The authors propose an intrinsic reward, which encourages an agent to learn valuable “skills”. The intrinsic reward is high if the changes in the environment is predictable for the specific ’skill’ and, if the changes in the environment are different for different

skills. The separate network is trained to predict environment changes for the specific skill, and its performance is used to calculate intrinsic reward value.

Hierarchical Actor-Critic by Levy et al. [15]. The authors propose multi-level hierarchical RL where environment states are used as sub-goals. I. e. all agents except the the lowest level agent have the set of environment states as their action space. The lowest level agent have the action space of the initial task. The lower-level agents are trained on the goal states, selected from the past experience similarly to the HER [1] approach.

Go-explore approach by Ecoffet et al. [8]. High-dimensional state is mapped into a low-dimensional space, with potentially high number of collisions, in order to group similar states together:

- without domain knowledge: state projected into a low-dimensional grid (low-resolution state image)
- with domain knowledge: vector of hand-crafted features from state image

Learning comprises two phases:

(1) Phase 1: Explore until solved
   (a) Choose a cell from the archive probabilistically (optionally prefer promising ones, e.g. cells with less number of visits) By explicitly storing a variety of stepping stones in an archive, Go-Explore remembers and returns to promising areas for exploration
   (b) Go back to that cell. In a resettable environment, one can simply reset the environment state to that of the cell. Most virtual environments are resettable (e. g. save/load state in empulator).
   (c) Explore from that cell (e.g. randomly for n steps)
   (d) For all cells visited (including new cells), if the new trajectory is better (e.g. higher score), swap it in as the trajectory to reach that cell
(2) Phase 2: Robustification. Robustify several found solutions (trajectories with high reward) into a deep neural network with an imitation learning algorithm. Imitation learning approach from "Learning from a single demonstration" is used.

Hindsight Experience Replay by Andrychowicz et al. [1]. A Multi-task RL agent gets current goal in addition to the observation. The goal is defined as the target state, that should be achieved. It is nearly impossible to accidental get the positive reward in the large state space. In Hindsight Experience Replay final states of the episodes are considered as target states for synthetic goals. The value function model is trained on both real goals and synthetic goals.

FuNs by Vezhnevets et al. [27]. The hierarchy of two agents is used: the master agent produces the goal vector and the worker agent produces actions. Worker agent is trained mostly on the intrinsic reward for moving in the goal direction inside state space.

GoCu by Bougie and Ichise [3]. The intrinsic curiosity reward for a state s_t defined as minus probability of reaching some pre-defined state, called goal. The reward approaches zero as probability of reaching a goal approaches one. Several random goal states are selected from the memory bank before start of the episode. The maximum probability of the probabilities of reaching the selected goals is used to calculate the intrinsic reward. The separate predictor is trained to estimate the probability of reaching a goal from the current state. Latent representations of the environment states are used, the states are embedded to the latent space by using variational auto-encoder.

SPTM by Savinov et al. [20]. A Multi-task RL agent gets current goal in addition to the observation. The goal is defined as a state of the environment. Two modules are used: the waypoint selection module and the locomotion module.

The waypoint selection consists from retrieval network R and the memory graph. The R outputs the measure of closeness between two observations, and is trained on binary classification task, where positive class is defined as a pair of observations with small action distance.

The memory graph's nodes are observations and an edge is set if two observations of different nodes are close enough according to R. During the execution phase, the node closest to the current observation and the node closest to the goal state are selected using R. The waypoint node is the node that lies on the shortest path between the current observation node and goal node and which is also close enough to the current observation according to R.

The locomotion module is implemented as the network L which takes current observation and waypoint observation and outputs the current action.

Both R and L are trained in self-supervised manner on the trajectories of the random agent.

Episodic Curiosity Through Reachability by Savinov et al. [21]. The authors propose to use minimum number of steps required to reach a state from some other state as the curiosity reward. The embeddings of the states with high curiosity reward are stored in the episodic memory and are used to estimate the reward. The compactor network is trained to estimate the number of steps between states and takes a pair of state embeddings as its input.

Automatic Goal Generation by Florensa et al. [9]. The authors consider the goal achievement task which can be described as the task to reach the specific embedding of an observation. They propose to use a goal generator implemented as a GAN which generate the goals which are similar in complexity to the previous iteration goals. The goals are generated to be increasingly challenging for the policy. In the experiments, the authors consider 2D embeddings of the Ant tasks as the goals.

Program Synthesis Guided RL by Yang et al. [29]. The authors propose three-component system: The Hallucinator g is a generative world model. The Synthesizer π uses Hallucinator to explore paths in the imaginary world which would satisfy the desired goal ϕ, the goal is explicitly defined by a user.

The Synthesizer produces a program p consisting from k components $\{c\}_1^k$. Each component c_t contains some condition β_t and some policy π_t which says to execute policy π_t until condition β_t holds. A user should provide a set of prototype components $\tilde{c}$, where $\tilde{c}$ is a logical formula which defines some useful subtask. The policy for $\tilde{c}$ is trained using RL.

Finally, the Executor executes p generated by π for N steps. If the desired goal ϕ is not satisfied, then the above process is repeated.

RIG by Nair et al. [19]. Goal-oriented RL scenario is considered. The goal is defined as a state to reach. The authors propose to use VAE to generate the latent representations of the states.

The reward is defined as the euclidean distance between the latent states. Minimizing this distance in the VAE latent space is equivalent to rewarding reaching states that maximize the probability of the latent goal. This is not true for normal autoencoders since distances are not trained to have any probabilistic meaning.

It is possible to sample new latent states from VAE, and, hence, sample new goals. By utilizing goal sampling it is possible to generate infinitely many goals and rewards for any observed state, which can be then used to train Q function estimator.

Abstract Models by Liu et al. [17]. Authors propose to to use manager-worker scheme where manager work in the space an abstract states which is substantially smaller than the space of actual states. Manager action space is the space of transitions between the abstract states. The new absract state is added if the discovered state is not assigned to any existing abstract state. The worker is

trained to reliably navigate between the pairs of abstract states. The exploration is performed by selecting an abstract state with low number of exploration attempts. The hand crafted-function to capture the grid cell of the agent is used to produce the abstract state.

Consciousness-Inspired Planning Agent by Zhao et al. [30]. The authors propose to use set-based representations of the observations instead of vector-based. The approach to produce state-based representations is taken from object detection work by Carion et al. [4]. The authors also propose to limit the objects used by world model by introducing the bottleneck, i. e. only a small subset of observation's objects is passed to the world model and returned by it. The returned subset of objects is integrated back to the observation to produce the predicted observation of the next state. The world model is used in the tree search planning algorithm to select most promising action of the agent.

RbExplore by Ugadiarov et al. [25]. The authors propose a version of Go-explore [8] exploration algorithm. They propose to use learned state embeddings instead of image upscaling. The state embeddings are learned in unsupervised manner using the closeness on the trajectory as similarity.

GMHRL by Chen et al. [5]. The authors propose a manager-worker framework, similar to PathRL, where the worker is trained on the state achievement task. The master is trained to generate potentially valuable states using the extrinsic reward. The buffer of observed states is used to train worker on the attainable goals. The algorithm is tested in the multi-task MiniGrid environment, where the current task is explicitly passed to the manager network.

Adversarially Guided Subgoals by Wang et al. [28]. Two-level hierarchy with manager and worker agents is used. Environment states are used as action space for the manager. Additionally, a discriminator net is trained to distinguish goals, generated by manager from the observed states. Discriminator provides additional GAN loss for the manager training.

Deep Hierarchical Planning by Hafner et al. [11]. The authors propose manager-worker agent scheme, which is similar to PathRL. Instead of raw states all components of the system operate on the latent state representations, produced by the PlaNet word model. The goal VQ-VAE is used to compresses state vectors into vectors of discrete codes. Manager selects a discrete code, that is then passed to the decoder, which restores it to the goal vector.

The goal vectors are restored to images by the additional decoder, trained along with the world model.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight Experience Replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

[2] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*, Feb. 2020. URL https://openreview.net/forum?id=Sye57xStvB. arXiv: 2002.06038.

[3] N. Bougie and R. Ichise. Skill-based curiosity for intrinsically motivated reinforcement learning. *Machine Learning*, 109:493–512, 2019.

[4] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020.

[5] D. Chen, Q. Yan, S. Guo, Z. Yang, X. Su, and F. Chen. Learning effective subgoals with multi-task hierarchical reinforcement learning. In *Scaling-Up Reinforcement Learning (SURL) Workshop. URl: http://surl. tirl. info/proceedings/SURL-2019_paper_10. pdf*, 2019.

[6] J. Choi, Y. Guo, M. Moczulski, J. Oh, N. Wu, M. Norouzi, and H. Lee. Contingency-Aware Exploration in Reinforcement Learning. In *International Conference on Learning Representations*, Mar. 2019. URL https://openreview.net/forum?id=HyxGB2AcY7. arXiv: 1811.01483.

[7] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *NIPS*, 1992.

[8] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return, then explore. *Nature*, 590(7847):580–586, Feb. 2021. ISSN 1476-4687. doi: 10.1038/s41586-020-03157-9. URL https://www.nature.com/articles/s41586-020-03157-9. Number: 7847 Publisher: Nature Publishing Group.

[9] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/florensa18a.html.

[10] D. Ha and J. Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

[11] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel. Deep hierarchical planning from pixels, 2022. URL https://arxiv.org/abs/2206.04114.

[12] S. Havrylov and I. Titov. Emergence of language with multi-agent games: Learning to communicate with sequences of symbols. In *NIPS*, 2017.

[13] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *ArXiv*, abs/1604.06057, 2016.

[14] H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The NetHack Learning Environment. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7671–7684. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/569ff987c643b4bedf504efda8f786c2-Paper.pdf.

[15] A. Levy, R. Platt, and K. Saenko. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryzECoAcY7.

[16] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018.

[17] E. Z. Liu, R. Keramati, S. Seshadri, K. Guu, P. Pasupat, E. Brunskill, and P. Liang. Learning abstract models for long-horizon exploration, 2019. URL https://openreview.net/forum?id=ryxLG2RcYX.

[18] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *ArXiv*, abs/1805.08296, 2018.

[19] A. Nair, V. H. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In *NeurIPS*, 2018.

[20] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SygwwGbRW.

[21] N. Savinov, A. Raichuk, D. Vincent, R. Marinier, M. Pollefeys, T. Lillicrap, and S. Gelly. Episodic curiosity through reachability. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkeK3s0qKQ.

[22] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[23] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *ArXiv*, abs/1907.01657, 2020.

[24] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[25] L. Ugadiarov, A. Skrynnik, and A. I. Panov. Long-term exploration in persistent mdps. In *MICAI*, 2021.

[26] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, and K. Kavukcuoglu. Strategic attentive writer for learning macro-actions. *ArXiv*, abs/1606.04695, 2016.

[27] A. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.

[28] V. H. Wang, J. Pajarinen, T. Wang, and J.-K. Kämäräinen. Hierarchical reinforcement learning with adversarially guided subgoals, 2022. URL https://arxiv.org/abs/2201.09635.

[29] Y. Yang, J. Inala, O. Bastani, Y. Pu, A. Solar-Lezama, and M. Rinard. Program synthesis guided reinforcement learning. *ArXiv*, abs/2102.11137, 2021.

[30] M. Zhao, Z. Liu, S. Luan, S. Zhang, D. Precup, and Y. Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *ArXiv*, abs/2106.02097, 2021.