Finding Structure in Deep Reinforcement Learning

Meng Song
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92037
mes050@eng.ucsd.edu

Abstract

The goal of this survey is to provide a comprehensive overview of discovering hierarchical and progressive structures in the context of modern reinforcement learning. Recent approaches and progress are reviewed and framed around two topics: hierarchical reinforcement learning (HRL) frameworks and the representation and discovery of subgoals. We explore extensively on relevant inspiring ideas, their developments and connections to other important problems in RL, and more broadly general AI.

1 Introduction

One of the hallmarks of human intelligence is the ability to compose basic components in novel ways. This capability reflects the structured nature of the world or at least how knowledge organizes our understanding of the world [1]. Our mental representation expresses scenes in objects and their interactions, objects in parts and attributes [2]. Our language is formed by constructing words into sentences, sentences into paragraphs [3]. We solve complex tasks by composing learned skills or familiar behavioral procedures in hierarchy with details of low-level control abstracted away [4]. For example, we achieve the task of attending this morning's computer vision class by bicycling to the campus, walking to the classroom, and listening to the lecture. All these forms of composition are driven by our cognitive mechanism and cultivate the emergence of abstract representations such as concepts, events, and intentions [5]. The limitless ways of creative combination of these building blocks is crucial to generalize our knowledge and thoughts to accommodate new situations [6, 7]. Yet a large gap still remains between human and artificial intelligence in terms of generalizable learning, in particular, for sequential decision-making problems. To cope with this challenge, it is natural to ask *what is the right way* to explicitly model the compositional hierarchy of goal-oriented behavior for generalization. Similar to a typical deep neural network [8, 9, 10, 11, 9, 12, 13] which enables *distributed representations* [14] by a hierarchical abstraction procedure and obtains generalization power from exponential combination of learned elements, the behavioral hierarchy is also studied and leveraged in reinforcement learning from a computational perspective.

Reinforcement learning [15] considers the problems where agents learn goal-directed behavior by trial and error in the environment. Being catalyzed by deep neural networks, modern RL methods have made significant progress, enabling agents to accomplish complex tasks such as simulated locomotion for navigation [16], robotic manipulation [17], Atari games [18], and Go [19]. However, current RL agents still suffer from considerable difficulties when facing:

1. **Sample efficiency**: Existing RL methods require huge amounts of interactions with environment, which are substantially more than a human would need. For example, despite combining all technical advances on DQN [18], Rainbow [20] requires 8 million frames of experience for training to achieve a human-level performance in most Atari games, which is approximately 80 hours of game experiences. Even without having to solve vision, training an agent to learn simple walking gait costs 15 to 20 CPU hours.

2. **Sparse rewards**: Because of the computational formulation, RL algorithms highly rely on reward signals to learn desired behaviors. However, in many real-world scenarios, the agent only experiences rewards occasionally or when the task is completed. And designing dense rewards precisely reflecting the task is impractical. Thus, most of the agent's exploration is futile, and success is by chance.
3. **Transferring experiences across tasks or environments**: Agents trained by deep RL methods are known to easily overfit to training environment. Slight change in dynamics or reward function (e.g., changing the mass of pendulum from 1 kg to 2 kg in pendulum swing-up task) can cause the learned policy fail to transfer and hard to adapt [21]. Leveraging neural networks as function approximator exaggerates this situation, resulting in an agent prone to be attracted by lazy local optima and weird environment regularities. This partly explains why RL is the only area in machine learning where it's socially acceptable to train on the test set.

Exploiting hierarchical structures in the space of controls or polices has been of longstanding interest in reinforcement learning. RL methods learning to behave on different levels of temporal abstractions are known as *hierarchical reinforcement learning* (HRL) [22]. In two-layer HRL architecture, high level controller learns to select not only one-step actions, but also low level policies performing in the action space, which are referred as *options*, *skills*, *macro-actions*, or *sub-policies*. To illustrate the benefits of behavioral hierarchy, we start by analyzing how HRL helps alleviating the aforementioned problems in RL:

1. **Structured exploration**: HRL enables the agent to explore with sub-policies rather than myopic primitive actions, making longer exploration possible in scaled-up state space. This capability of reaching non-local environment configurations is critical for building the world model, thus helps improving sample efficiency. Chaining learned skills [23] also quickly focuses exploration and modeling on paths near subgoals, therefore increases agent's chances of being rewarded and discovering critical transitions.
2. **Transferable behavior modules**: Unlike directly transferring complex skills, sub-policies learned in HRL can be efficiently transferred to a related task, considering that they have already been constrained in small regions of state space.

Besides proposing promising solutions to these fundamental issues in RL, HRL also facilitates long-term planning and reasoning [24], and *successfully solves a lot of complex tasks otherwise unapproachable by single flat policy*. Typically, evaluations are performed on continuous control tasks requiring an agent to achieve goals by navigation, which involves locomotion skills and basic object interactions, e.g. Ant Maze, Ant Push, and Swimmer Navigation.

For concreteness, Options framework [22] presents a formal definition of HRL by incorporating it into MDP formalism where the high-level controller can directly operate over sub-policies without concern for the details of their execution. However, this general formulation does not enforce any fixed form to the separation of low-level policies, and providing strategies to automatically induce *semantically distinct* sub-polices is challenging. For this reason, constructive efforts have been made in the field to discover sub-polices [25, 26, 27]. In most cases, the resulting sub-policies either entail simple sequential action patterns or lack any intentions behind. In other words, the agent decomposes its behavior in strict accordance with policy structure, in which procedure the structure of state space is out of awareness.

In contrast to leaving decomposition motivation and mechanism unclear, Feudal framework explicitly defines how high-level controller modulates low-level policies, i.e., high level controller (*manager*) generates subgoals, and *worker* executes policies to achieve them. The central concept *subgoal* corresponds to a group of states which acts as waypoints to guide the agent towards completing the overall goal. Subgoal bridges the manager and worker and naturally gives rise to intrinsic rewards. By centering around subgoals, the focus of structure discovery is switched from temporal abstraction to state space abstraction, which from cognitive science perspective, corresponds to a transition from dividing concrete behaviors to extracting abstract intentions. The design of Feudal framework aligns well to the observation that when humans are facing a complex task that requires long-term planning or deliberation, they usually set up subgoals within their knowledge space, and solve the whole task by divide and conquer. In this sense, subgoals can be viewed as scaffolding for building the optimal behaviors.

Although subgoals help decompose a hard task into manageable sub-tasks, subgoal representation and discovery are still fundamental and unresolved problems. By learning an embedding of the state space as subgoal representations, a large number of methods has made progress on a range of tasks, especially in continuous control domain. Most of them can be grouped according to whether a subgoal is mapped to a set of target states or some specific features of the state space. Since the agent is optimizing its policy during the learning process, self-discovered subgoals should progressively increase the learning difficulty to match the agent's capability and expand the knowledge boundary efficiently. This motivates us to inspect the problem of subgoal discovery in the context of curriculum learning. The discussion will proceed from the agent's exploration in goal space to its active modification of environments.

Generally, in this survey we study how AI's capability for behavior generalization and self-motivated learning are improved by biasing reinforcement learning methods towards hierarchical and developmental structures. And how this problem is strongly connected to a wide range of key topics in reinforcement learning and general AI. The works cited in this survey are by no means an exhaustive list of efforts on answering these fundamental questions throughout the history, but aim to provide a representative cross-section of the breadth of inspiring insights and seminal ideas scattered.

The rest of this survey is structured as follows. We set out by a brief overview of reinforcement learning framework and techniques in Section 2. Then we come to focus on a general HRL design, i.e. the Options framework in Section 3. Section 4 is devoted to the definition and discussions around the central concept subgoal and its associated hierarchical architecture, i.e. the Fedual framework. Section 5 extends the idea of subgoal by establishing and developing its relationships to curriculum learning. Finally, we conclude the survey with suggestions on future directions in section 6.

2 Background: reinforcement learning

2.1 Problem setting

Reinforcement learning is a framework where an agent learns to perform decision making or control by interacting with the environment and getting feedback over periods of time [15]. At each discrete time step t, the agent receives an observation o_t of the state of environment s_t, and accordingly selects an action a_t to take. After the agent acts on a_t, it will receive a reward signal r_{t+1} at the next time step and affect the environment's transition to the next state s_{t+1} (Fig. 1).

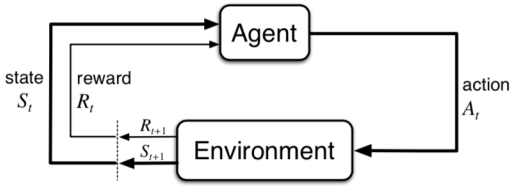

Figure 1: The agent–environment interaction in a Markov decision process

A *state* s contains the complete information about how the environment works, whereas an *observation* o could be a partial description of s which is constrained by agent's perception.

> In the literature, people often use s in place of o, however it is not technically rigorous since the agent does not have access to the real state of the world in most of the cases. This review will respect the notations in the cited works, and expect the readers to infer the precise meaning of the notation from the context.

In aforementioned interaction loop, the environment can be specified by a tuple $(S, A, P, R, \rho_0, \gamma)$, with S the state space, A the action space, ρ_0 the initial state distribution, and $\gamma \in [0, 1]$ a discount factor. $R : S \times A \times S \to \mathbb{R}$ is a task-specific reward function with $r_t = R(s_t, a_t, s_{t+1})$. P defines the environment *dynamics* which can be written as a transition probability distribution $P(s_{t+1}|h_t, a_t)$ in the stochastic case. P and R together specify the environment's behavior, thus constitute a *model* of the environment.

Generally, the environment state s_{t+1} depends on history $h_t = (s_0, \cdots, s_t)$. However, almost all systems RL is dealing with can be simplified to having transitions satisfying the Markov property. Specifically, $P = P(s_{t+1}|s_t, a_t)$, i.e. current state is sufficient statistics of the future. An environment in this setting is called *Markov Decision Processes (MDPs)* [28].

On the other hand, the agent's behavior is formally specified by its *policy* π, which is a mapping from states s_t to actions a_t. A stochastic policy is usually denoted by a probability distribution $\pi(a_t|s_t)$. To measure the agent's policy, we define a *return* as the discounted sum of rewards over a n step trajectory (or called *episode* or *rollout*) τ:

$$R(\tau) = \sum_{t=0}^{n-1} \gamma^t r_t \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, \cdots, s_{n-1})$ is generated by sampling actions according to a particular policy. The discount factor γ determines how immediate rewards are favored relative to long term rewards.

The probability of trajectory τ is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{n-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2}$$

In MDP framework, the agent's goal is to find an optimal policy π^* which maximizes the expected cumulative reward in the long run. This can be defined formally by maximizing expected return $J(\pi)$ of policy π, where $J(\pi)$ integrates over all possible future trajectories τ produced by π:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3}$$

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)] \tag{4}$$

Although for the most part, the MDP framework is flexible and general enough to characterize the problem of learning goal-directed behavior from interactions, reinforcement learning can be taken beyond MDPs [15].

2.2 Value functions

Value functions are central concepts to reinforcement learning problems. The recursive relationships derived from value functions lead to important properties in estimating optimal policies, which are known as *Bellman equations*.

By definition, there are two types of value functions which estimate how good a state, or state-action pair is under a given policy in terms of future rewards. To see this clearly, one can draw correspondences to the value of vertices and edges in the graph representation of MDPs.

2.2.1 Value function

The value function of state s under policy π describes how much future rewards the agent can expect to receive when it starts from state s and acts according to policy π. This is formally defined as:

$$V^{\pi}(s) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s] \tag{5}$$

where state s_0 indicates the starting point of trajectory τ.

Bellman equation for $V^{\pi}(s)$ is:

$$V^{\pi}(s) = \underset{a\sim\pi, s'\sim P}{E}[r(s, a) + \gamma V^{\pi}(s')] \tag{6}$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s, a)$, meaning that the next state s' is sampled from environment transition distribution P. Notation $a \sim \pi$ indicates that action a is sampled from policy $\pi(\cdot|s')$.

The optimal value function gives the expected return if the agent starts from state s and acts according to the optimal policy π^*, which amounts to the maximum value for state s achievable by any policy:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{7}$$

Bellman equation for $V^*(s)$ is:

$$V^*(s) = \max_{a} \underset{s' \sim P}{E}[r(s,a) + \gamma V^*(s')] \tag{8}$$

When the environment model is known, equation (8) can be used as the update rule to calculate the optimal value for all states in standard *value iteration algorithm*.

2.2.2 Action value function

The action value function of state action pair (s, a) under policy π gives the expected return when an agent starts from state s, taking action a, which may not come from the policy, and then acts according to policy π. It could be written as:

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{E}[R(\tau)|s_0 = s, a_0 = a] \tag{9}$$

Bellman equation for $Q^{\pi}(s,a)$ is:

$$Q^{\pi}(s,a) = \underset{s' \sim P}{E}\left[r(s,a) + \gamma \underset{a' \sim \pi}{E}[Q^{\pi}(s',a')]\right] \tag{10}$$

where action a' is taken from state s'.

Estimating value functions based on (6) or (10) is a major building block of *policy iteration algorithm*. This algorithm alternates between policy evaluation and greedy policy improvement, and is guaranteed to converge to the optimal policy. In most of the cases, both of these two steps rely on value functions heavily. Policy gradient algorithms in modern RL share a similar structure.

The optimal action value function gives the expected return if the agent starts from state s, takes action a, and then always follows the optimal policy π^*:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{11}$$

Bellman equation for $Q^*(s,a)$ is:

$$Q^*(s,a) = \underset{s' \sim P}{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')] \tag{12}$$

Equation (12) gives rise to another form of the value iteration algorithm.

If $Q^*(s,a)$ is given at state s, we can directly get the optimal action a^* by:

$$a^*(s) = \arg\max_{a} Q^*(s,a) \tag{13}$$

Furthermore, if $Q^*(s,a)$ is available for all $s \in S$, we have the optimal policy π^*.

2.2.3 Connections between V and Q

From the definitions, we can directly derive the key connections between the value function V and the action value function Q:

$$V^{\pi}(s) = \underset{a \sim \pi}{E}[Q^{\pi}(s,a)] \tag{14}$$

$$V^*(s) = \max_{a} Q^*(s,a) \tag{15}$$

$$Q^{\pi}(s,a) = r(s,a) + \gamma \underset{s' \sim P}{E}[V^{\pi}(s')] \tag{16}$$

$$Q^*(s,a) = r(s,a) + \gamma \underset{s' \sim P}{E}[V^*(s')] \tag{17}$$

2.2.4 Advantage function

In RL algorithms, we often care more about the action's relative advantage than its absolute value, or more precisely, how much better is the specific action a than randomly selecting an action according to π. This is expressed by:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{18}$$

2.3 RL algorithms

Modern RL algorithms have been extended to solve increasingly complex tasks in a variety of domains, such as game playing, locomotion, and manipulation, in large part because of using expressive neural networks as function approximators. In this section, we will briefly introduce several popular model-free deep RL algorithms to equip the readers with basic knowledge necessary to understand later discussions.

2.3.1 Policy gradient algorithms

Model-free RL algorithms branch into two families, policy gradient and value-based algorithms, based on whether the policy is directly optimized or induced from value functions.

The central idea of policy gradient algorithms [29] is straightly optimizing the policy by gradient ascent to encourage the actions leading to high return. The policy gradient is calculated by forming an estimator of the gradient of expected return from sample trajectories. We will first derive the vanilla policy gradient algorithm, then extend it to more general cases.

2.3.1.1 Vanilla policy gradient algorithm

We consider a stochastic policy π_θ parameterized by θ. The objective function (4) is thus written as:

$$J(\pi_\theta) = \int_{\tau} P(\tau|\pi_\theta) R(\tau) = \underset{\tau\sim\pi_\theta}{E}[R(\tau)] \tag{19}$$

To optimize π_θ along its gradient, we can update θ by:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{20}$$

The policy gradient $\nabla_\theta J(\pi_\theta)$ can be analytically derived as:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\Big[\sum_{t=0}^{n} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\Big] \tag{21}$$

The outer expectation is further estimated by the sample mean of trajectories $D = \{\tau_i\}_{i=1}^{N}$ collected under policy π_θ:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{n} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i) \tag{22}$$

2.3.1.2 Actor-critic architecture

The problem of vanilla policy gradient in (22) is that the future reward $R(\tau_i)$ is estimated from a single trajectory while omitting other possible futures. It is unbiased but with high variance. To remedy this, we replace $R(\tau)$ in expression (2.3.1.1) with Φ_t, and have the general form policy gradient:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\Big[\sum_{t=0}^{n} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t\Big] \tag{23}$$

Now $\nabla_\theta J(\pi_\theta)$ can be interpreted as the maximum likelihood of the policy weighted by Φ_t. Φ_t can take different forms, such as $\sum_{t'=t}^{n-1} R(s_{t'}, a_{t'}, s_{t'+1}) - b$ (b is an arbitrary baseline of the cumulative reward), $Q^{\pi_\theta}(s_t, a_t)$, and $A^{\pi_\theta}(s_t, a_t)$. Among them, $A^{\pi_\theta}(s_t, a_t)$ yields the lowest variance [30]. It can be approximated involving only V function:

$$A^{\pi_\theta}(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\phi^{\pi_\theta}(s'_t) - V_\phi^{\pi_\theta}(s_t) \tag{24}$$

The actor-critic architecture builds on the standard policy gradient algorithm by adding a *critic* to approximate value function V or Q. The algorithm first samples $\{(s, a, s', r)\}$ under current policy,

then the critic updates the value function, finally the *actor* updates the policy along the gradient direction suggested by the value function according to (23). In deep RL, the value function and policy are parameterized by separate neural networks. A3C [31] executes multiple parallel actors trained on their own instances of the environment, thus accelerates and stabilizes training a lot.

2.3.2 Value-based algorithms

In value-based reinforcement learning methods, the optimal action is directly selected based on the value function approximator, instead of having an explicit policy function.

2.3.2.1 DQN

Deep Q-network (DQN) [18] was one of the first breakthroughs in applying RL methods to learn successful policies end-to-end from high-dimensional sensory inputs approaching real-world complexity. DQN was shown to be able to master a diverse range of Atari games to superhuman level.

One core design of DQN is the usage of neural networks in classic Q-learning. Motivated by (13), Q-learning learns the optimal Q function and selects the action yielding highest value evaluation as its best policy. Based on Bellman equation (12), the recursive structure is exploited as loss function:

$$L(\phi, D) = \underset{(s,a,r,s')\sim D}{E}\left[\Big(Q_\phi(s,a) - \big(r(s,a) + \gamma \max_{a'} Q_\phi(s',a')\big)\Big)^2\right] \tag{25}$$

which measures how closely $Q_\phi(s,a)$ satisfies the Bellman equation.

$Q_\phi(s,a) - \big(r(s,a) + \gamma \max_{a'} Q_\phi(s',a')\big)$ is the *temporal difference (TD) error*, where $y \triangleq r(s,a) + \gamma \max_{a'} Q_\phi(s',a')$ is the regression *target*. In TD learning, the target is set to be n step ahead existing estimate rather than complete return at the end of the episode, thus is known as *bootstrapping*.

Approximating $Q_\phi(s,a)$ using large non-linear function estimator such as a neural network will cause the learning diverge or oscillate catastrophically. To address this problem, two key techniques were introduced which have since been successfully adopted by many subsequent deep RL algorithms:

1. *Replay buffer*: In online Q learning, samples (s,a,r,s') are consecutively generated from behavior policy ρ's exploration in the environment. However, optimization algorithms usually assume that the samples are i.i.d. Moreover, learning on-policy should be avoided so that the parameters are decoupled from the data they are trained on. To tackle these issues, DQN stores all transitions (s,a,r,s') encountered during training in the so-called replay buffer (annotated as D in (25)), and samples random batch from it when performing updates.
2. *Target network*: The target in (25) is problematic as it depends on parameters ϕ we trained on, thus varies a lot during training. In order to make the target more stable, a separate network Q_{targ} is introduced to parameterize Q in the target, which is called target network. Q_{targ} should change at a slower pace than the main network Q_ϕ. Common practice is either periodically copying the weights of Q_ϕ to Q_{targ} or using a polyak averaging of Q_ϕ's weights once Q_ϕ is updated.

By putting them together, we can optimize main network Q_ϕ according to:

$$\phi \leftarrow \phi - \alpha \sum_{(s,a,r,s')} \Big(Q_\phi(s,a) - \big(r(s,a) + \gamma \max_{a'} Q_{targ}(s',a')\big)\Big)\nabla_\phi Q_\phi(s,a) \tag{26}$$

2.3.3 DDPG

The requirement of computing the maximum over actions makes DQN only suited to solving RL problems in discrete action space. By contrast, Deep Deterministic Policy Gradients (DDPG) [32] is devised to learn in continuous action spaces.

DDPG adapts DQN by embedding $Q(s,a)$ in an actor critic architecture as a critic. The actor is a deterministic policy $\mu_\theta(s)$ which produces the action maximizing $Q_\phi(s,a)$. In other words, policy network μ_θ acts as an approximate maximizer. Assuming that Q-function is differentiable with respect to action, we can update $\mu_\theta(s)$ by gradient ascent with respect to θ to solve:

$$\max_\theta \underset{s\sim D}{E}[Q_\phi(s, \mu_\theta(s))] \tag{27}$$

Specifically,

$$\theta \leftarrow \theta + \beta \sum_i \nabla_\theta \mu_\theta(s_i) \nabla_a Q_\phi(s_i, a) \tag{28}$$

Optimization of $Q(s, a)$ still follows (26). The only difference is that the optimal action in the target is computed according to policy network μ_θ:

$$y = r(s, a) + \gamma \max_{a'} Q_{targ}(s', \mu_\theta(s')) \tag{29}$$

3 Options framework

3.1 Definition

The most well-known formulation for HRL is probably the Options framework [22]. The central concept $options$ are proposed for generalization of primitive actions to include temporally extended courses of action over a period of time. Formally, options are defined as (I, π, β) including a policy $\pi : S \times A \rightarrow [0, 1]$, a stochastic termination condition $\beta : S^+ \rightarrow [0, 1]$, and an initiation set $I \subseteq S$. An option could be activated at state s iff. $s \in I$. If an option is taken, the agent will act in the environment according to π until β is met. Options in real life could be opening a door or having a dinner which normally involves long sequence of primary actions described by joint torques and muscle twitches.

By augmenting its action space with options, an agent can now choose to perform not only primary actions, but also options based on current observation. This forms a policy over options, which can fit to standard RL methods by simply treating options as actions.

In practice, options could either be provided in sketch [25] or learned from human demonstrations, and are well suited to be deployed for solving complex tasks with innate hierarchies or subgoals. Prominent examples could be found where AI is learning to play complex strategy games such as Starcraft II [33] or Minecraft [34]. With mastered skills, large and complex state and action spaces could be efficiently reduced, which enables quick strategic decision making in high-level controller. When training to play against competitors, each sub-policy can have its own specific reward function which helps flexible response.

3.2 Option discovery

Previous works have attempted to automatically discover *meaningful options* in a variety of ways and have provided encouraging successes. However, most of the behavior primitives are induced by task structures [26, 27]. Otherwise, the sub-policies tend to degrade to trivial solutions of either only one option is learned to solve the whole task or option is changed at every time step.

This section will introduce several representative but very different heuristics to discover meaningful behaviors, and discuss their advantages and applicable scenarios.

3.2.1 Emergence of options from behavior switching

One simple and direct way to temporally abstract a sequential decision making process is to view it as multi-stage behavior switching problem. To develop this idea, STRategic Attentive Writer (STRAW) [27] learns macro-actions by building implicit action plans and choosing to either update current plan or commit to it. Being different from standard HRL setting, the low-level controller does not generate sub-polices, i.e., it does not produce action from every observation, but instead plan a sequence of actions from one informative observation. This lookahead allows the agent to allocate computation only on key moments.

Let A be the number of possible actions and T the maximum time horizon of the plan. The dynamic action plan at time step t can be represented by two matrices:

- **Action-plan** $\mathbf{A}^t \in R^{A \times T}$ describes the plan of future actions conceived at time t. The first column of $\mathbf{A}^t$ corresponds to a probability distribution over actions that the agent will take at time t.

- **Commitment-plan** $\mathbf{c}^t \in R^{1\times T}$ is a row vector where the first element represents a Bernoulli distribution of binary variable g_{t+1}. If $g_{t+1} = 1$, the action-plan $\mathbf{A}^{t+1}$ will be updated at the next time step. Otherwise, the agent will commit to the current plan, i.e., $\mathbf{A}^{t+1} = \rho(\mathbf{A}^t)$, where ρ reflects the advancement of time by shifting the matrix left for one column. A **macro-action** is defined as a sequence of output actions $\{a_{t_1}, \cdots, a_{t_2-1}\}$, where $g_{t_1} = g_{t_2} = 1$ and $g_{t'} = 0, \forall t_1 < t' < t_2$.

The usage of this dynamic action-plan is established on the assumption that one observation contains sufficient information to generate a sequence of actions. However, the complexity and length of a macro-action could vary dramatically in reaction to the situation change. Therefore, the read and update of action-plan should only focus on the part of the plan where current observation is informative of desired actions. This attention mechanism is implemented by applying a grid of $K \times A$ of one-dimensional Gaussian filters to $\mathbf{A}^t$ over the temporal dimension. The grid position, stride and standard deviation of Gaussian filters are parameterized by a vector ψ_t^A, which can be regressed from the feature representation z_t of current observation x_t:

$$\psi_t^A = f^{\psi}(z_t) \tag{30}$$

Given z_t, the network can also produce an action patch $\mathbf{q}_t$ by comparing the observation and the old plan. The update to the action-plan $\mathbf{A}^t$ is then created by scaling and shifting $\mathbf{q}_t$ according to ψ_t^A.

The commitment-plan $\mathbf{c}^t$ is updated at the same time as $\mathbf{A}^t$. Unlike the additive update to $\mathbf{A}^t$, $\mathbf{c}^t$ is overwritten completely.

STRAW has been demonstrated to be able to learn meaningful action patterns and reactive strategies in 2D maze, Atari games, and text. For instance, in 2D maze navigation, corners and areas close to junctions are shown to correspond to locations where macro actions terminate and re-planning happens. In game Frostbite, high-level actions such as jumping from floe to floe and picking fish are learned. It is also observed that in game Amidar, when an enemy blocks the way, STRAW drastically changes its plan and retreats to the left. Beyond learning in RL setting, STRAW can be viewed as a general sequence prediction model to capture data with complex structure. For example, in character prediction task, STRAW is capable of learning macro-actions corresponding to common n-grams.

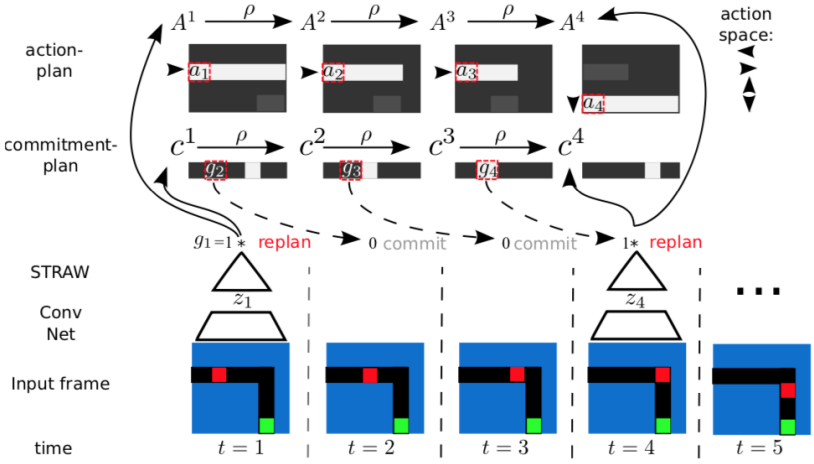

Figure 2: Illustration of STRAW playing a 2D maze navigation game

3.2.2 Meta learning of hierarchical policies

MLSH [26] formulates the learning of a two-layer hierarchy of policies in meta learning setting, where meaningful sub-policies are naturally discovered by training over a distribution of related tasks. The model follows the Options framework and contains a set of sub-policies shared across tasks, which are switched between by a task-specific master policy on top. The learning objective hypothesizes that a good hierarchy should find a set of sub-polices that allow quick learning of master policy when it is transferred to new tasks.

Note that in this survey, we make the distinction between the terms *task* and *goal* as: **Tasks** *may* have different MDP dynamics, whereas **goals** can only differ in reward functions but not environment transitions. For example, a maze task could change the maze's configuration, however, goals in a maze normally refer to different locations.

To achieve this objective, the underlying structure of the related tasks is exploited so that:

- The **sub-policies** represent shared sub-tasks. An optimal set of sub-polices discovered should be both robust and diverse.
- The **master policy** captures how to compose sub-tasks and implements a task-specific switch. That is, transferring this hierarchical agent to a new task can be done by solely updating the master policy on how to schedule the sub-policies.

Formally, MLSH consists of a stochastic *master policy* parametered by θ, and K *sub-policies* $\phi_1, \ldots, \phi_K$. the agent is trained on M tasks during its lifetime. For iteration of a specific task, the agent starts with a random θ, then samples task M from a distribution P_M over MDPs. Sub-policies ϕ carried from previous iterations are optimized by interacting with the environment for T steps over multiple episodes, and R is the total return. For each N time steps in an episode, the master policy takes an action to activate a sub-policy by sampling an index $k \in \{1, 2, \ldots, K\}$. (Fig. 3)

The overall learning objective is defined as:

$$\max_{\phi} E_{M \sim P_M, t=0 \ldots T-1}[R] \tag{31}$$

To optimize (31), the learning algorithm iteratively alternates between two periods:

- Warm-up period: optimize θ.
- Joint update period: optimize θ and ϕ jointly.

The reason for setting up a warm-up period is that ϕ should only be updated when θ is near optimal.

Comparing with other HRL methods, the architecture of MLSH has several notable advantages:

- There are no gradients between the master and sub-policies. They communicate solely by an one-hot sub-policy index. The simplification of communication can also be seen in the design of Feudal framework [35].
- The lack of gradient communication also allows MLSH to be agnostic to learning methods.
- Both the master and sub-polices are allowed to observe the environment and receive rewards, which makes learning easy.

Rich experiments have been conducted to show that diverse sub-policies can be automatically learned through MLSH. For example, directional movement primitives such as move up, right, and down have been discovered where an Ant is trained on a distribution of mazes. Another interesting scenario is that an agent is rewarded to reach one of two randomly placed points. Each point has its own fixed color and the agent does not know which one is the goal. After training, sub-policies of movements towards each point are automatically learned to decompose and solve this problem.

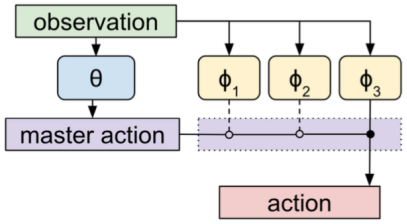

Figure 3: Architecture of MLSH, where $K = 3$

3.2.3 Representation learning of options

A line of recent works [36, 37] has attempted to embed sub-policies as latent variables or add a latent layer in between to facilitate high-level planning, efficient modeling, and exploration.

As a specific example, SeCTAR [37] implements the options framework by a trajectory-level *variational autoencoder (VAE)* [38], which learns to generate and predict behaviors. In addition, a world model operating on the trajectory level is induced from this VAE. An MPC controller planning on the latent space of VAE takes the role of the master policy.

The core VAE is comprised of three components:

- An **encoder** $q_\phi(z|\tau)$, which embeds trajectory τ into a latent variable z.
- **State decoder** $p_{\theta_{SD}}(\tau|z)$, which can decode z directly into a trajectory.
- **Policy decoder** $p_{\theta_{PD}}(a|s,z)$ representing a latent-conditioned policy, which can generate a trajectory by acting in the environment sequentially.

where trajectory $\tau = [s_0, s_1, \ldots, s_T]$ is a sequence of states, which are taken from a *short* segment of an episode.

The state decoder can be thought of as a model to predict the outcomes of the policy. This predictive model is not built at the state-action transition level, which makes the model building substantially easier.

On the other hand, to train the VAE, the policy decoder is encouraged to imitate the state decoder's behavior, generating a trajectory that matching trajectory predicted by the model.

The overall objective function maximizes the likelihood of trajectory $p(\tau)$ as well as the consistency between the state decoder and policy decoder:

$$\begin{array}{ll} \max & \log p(\tau) \\ s.t. & \mathbb{E}_{q_\phi}\left[D_{KL}\left(p_{\theta_{PD}}(\tau|z) \| p_{\theta_{SD}}(\tau|z)\right)\right] = 0 \end{array} \tag{32}$$

The state decoder allows us to perform model-based planning using model predictive control (MPC). The latent space is just the action space of MPC controller, where one can sample a random z and simulate its trajectory by the state decoder. Trajectories for training are collected by an explorer starting from the states visited by MPC controller and exploring diversely by *maximum entropy*.

We summarize the advantages of SeCTAR as following:

- A continuous latent space is learned for the trajectories rather than a discrete set of skills, which makes SeCTAR applicable to continuous control tasks.
- SeCTAR enables model-based long-horizon planning.
- Leveraging the properties of Gaussian distribution in VAE, the latent space organized trajectories functionally. One can interpolate the latent variables and visualize the corresponding trajectories.
- Training a master policy is not needed.

4 Feudal framework and subgoals

4.1 Definition of feudal framework

Feudal framework [39] is a powerful and efficient HRL design where different levels of hierarchy within an agent communicate via explicitly defined *subgoals*. At high level, a *manager* operates to set abstract subgoals, whereas a *worker* is employed to achieve them at a level below. The manager makes high-level reasoning at a lower temporal resolution than the worker behaves at. Two key advantages arise in this *subgoal-centric* architecture:

- **Decoupling**: It decouples end-to-end learning across levels into two parts: goal setting and goal achievement, which clearly separate the duties of manager and worker. The only shared actionable information between manager and worker is a latent goal space.

- **Flexibility**: First, how to achieve subgoals is not specified by the manager. Second, the worker can choose to interact with the environment or only receive intrinsic rewards elicited from subgoals.

In Feudal framework, an agent's behavior can be viewed as chaining a set of sub-trajectories directed by intrinsically generated goals.

4.2 Definition of a general goal space

In the classic reinforcement learning paradigm, the agent's *goal* is defined as maximizing *task-dependent* cumulative rewards on a MDP. For example, in a 2D maze, an agent could be asked to achieve a goal location in the upper-left corner guided by reward function r_1, and a different goal location in the right-bottom corner described by another reward function r_2. Therefore, such goals constitute a *goal space* $\mathcal{G}$, where each goal $g \in \mathcal{G}$ corresponds to some extrinsic reward function $r(g)$. Based on $\mathcal{G}$, the notion of value functions $V(s)$ and $Q(s, a)$ can be generalized to $V(s, g)$ and $Q(s, a, g)$, and so can their optimal counterparts. For example, general value function $V(s, g)$ now caches a chunk of knowledge about the utility of any state s in achieving a given goal g [40]. Thus, $V(s, g)$ can be viewed as the knowledge representation of how to evaluate or control a specific aspect of the environment. That is, the progress from subgoal s towards goal g.

Accordingly, the agent's capability is expected to go beyond achieving a single overall goal. To explore the whole goal space, now the agent is required to learn a *goal conditioned policy* $\pi(a|s, g)$ that can achieve multiple goals in the environment. This capability of mastering broadly applicable and general-purpose skills is not only a pursuit of general AI, but also aligns with the practical requirement for an agent to fulfill a wide range of user-specified goals at test time.

However, should the agent be rewarded only when it sets the goals as achieving external tasks? Indeed, this strong association between goals and tasks excludes the rich supervision signals innate in the environment dynamics itself, and poses the challenge of sparse extrinsic rewards to the agents. To remedy this issue, the goal space $\mathcal{G}$ should be further generalized to incorporate the **manipulation of environment**, i.e., $\mathcal{G}$ could be extended to the observation space. Concretely, consider the *sensori-motor stream* $X = (o_1, a_1, o_2, a_2, \cdots, o_t, a_t, \cdots)$ produced by the agent-environment interaction, a more general objective for the agent is to learn to **predict and control** this stream [41], or the future. For instance, in Ms. PacMan, an agent could learn to maximize the accumulated number of pellets and minimize the number of ghosts it observes by actively approaching or avoiding them. It is hypothesized that an agent capable of flexibly controlling its future is certainly competent for maximizing extrinsic rewards induced by any task. This is because the future is a very rich space, even to make a small change to a pixel may require the agent to master complex and immense behaviors, and more importantly much deeper understanding of the environment dynamics. And the behaviors learned from reaching one goal are likely to recur for many other goals, which suggests that the vastness and richness of goal space would endow the agent with strong generalization ability.

4.3 Definition of subgoals

The notion *subgoal* introduced in Feudal framework has richer meanings and broader reach far beyond serving a specific architecture of HRL. In essence, subgoals are waypoints to reach a general goal defined above. With the help of subgoals, the agent does not need to perform global optimization on the policies over the entire action-state space or all sampled trajectories. Subgoals could provide directions or intermediate targets to guild the agent's behavior. In the cases where a measure could be defined in the goal space, an agent can even use subgoals to measure its progress to the ultimate goal.

In summary, by exploiting the structure of task space, an agent can employ subgoals to decompose a hard and complex task into a set of easy and simple subtasks, then solve them by divide and conquer. This could be done by a single agent with hierarchical controller, a single agent with flat policy but regularized by subgoals, or even multi-agents acting in the same environment.

4.4 Subgoal representation

Depending on the scope of goal space, subgoals are endowed with different interpretations, thus their representations fall into two categories:

In the first case, **subgoals are defined in terms of states**, e.g. the goal space $\mathcal{G} \subseteq \mathcal{S}$. Several useful representations include:

- Subgoal is a target state $s_T \in \mathcal{S}$ [42, 43, 16].
- Subgoal corresponds to a subset of states [44, 45].
- Subgoal is a direction vector in state space [35].

The closeness of state space and goal space allows one to exploit the common structure between states and goals. For example, Universal Value Function Approximators (UVFA) [40] are designed to learn a generalized value function for any state-goal pairs.

In the second case, **subgoals are defined in terms of the observation space**. That is, *a subgoal focuses on controlling some aspect of the future* [46]. For example, in rich visual environments, an agent's subgoals could be to manipulate some factors of the visual content via its actions [41, 47]. From a HRL perspective, the meta controller tells the sub-controller which aspects of the input observation it should control, and the sub-controller receives intrinsic rewards for successfully changing the content and as well as the extrinsic rewards from the environment for learning fine-grained behaviors. Based on the abstraction level, subgoal representations in this case can be broken into two categories:

- **Pixel control**: Subgoals are represented in raw image space. The agent is trained to learn a sub-policy to maximally changing the pixels in a given patch between two consecutive frames.
$$r^{int}(k) = \eta \frac{\left\| \boldsymbol{h}_k \odot (\boldsymbol{s}_t - \boldsymbol{s}_{t-1}) \right\|^2}{\left\| \boldsymbol{s}_t - \boldsymbol{s}_{t-1} \right\|^2} \tag{33}$$
where $\boldsymbol{h}_k$ selects the given (k^{th}) patch and s_t denotes the pixels in frame t. The change in values of pixels in a patch is measured relative to the entire screen.
Experimental results on Montezuma's Revenge demonstrate that by controlling pixels around rewarding objects, such as a key, the agent is driven to reach that target region.
- **Feature control**: Subgoals are represented in feature space. The agent aims to control some desired features and ignore the others, which can be implemented by maximizing the activation of specific feature map (a group of neurons) in the second convolutional layer of CNN.
$$r^{int}(k) = \eta \frac{\left\| f_k(\boldsymbol{s}_t) - f_k(\boldsymbol{s}_{t-1}) \right\|}{\sum_{k'} \left\| f_{k'}(\boldsymbol{s}_t) - f_{k'}(\boldsymbol{s}_{t-1}) \right\|} \tag{34}$$
where $f_k(\cdot)$ denotes the mean over activation values in the k^{th} feature map. And the induced intrinsic rewards are normalized.

In the following sections, we will review several prominent subgoal representations and their roles in HRL.

4.4.1 Raw states as subgoals

In continuous control setting, the state space is usually low dimensional, fully observed, and close to the right representation level supporting the underlying dynamics. Therefore, directly using states as subgoals in their raw form becomes possible. This simple definition of subgoals significantly simplifies the architecture design of Feudal framework and accelerates the training of sub-polices as there is no need to learn goal representations.

HIRO [42] embodies this idea by a concise off-policy architecture, in which both manager and worker are trained by DDPG-based algorithms. For each period of c time steps, the worker produces a **goal state** $q_t \triangleq s_t + g_t$ indicating the objective state to achieve in current period. And the worker would try to perform a course of actions causing observation s_{t+c} closely matching q_t. For example, in the task of Ant Push, q_t corresponds to the ant's target pose.

Note that the manager's **action** g_t is directly defined as changes in the state. It is either picked from the manager's policy or a goal transition function $h(s_{t-1}, g_{t-1}, s_t) = q_{t-1} - s_t$ to keep the absolution location of goal q_{t-1} proposed at the previous time step unchanged.

Similarly, the worker's intrinsic reward is parameterized according to L_2 distance between s_{t+1} and goal state q_t, measuring current progress towards the goal.

$$r\left(s_t, g_t, a_t, s_{t+1}\right) = -\left\|q_t - s_{t+1}\right\|_2 \tag{35}$$

Although off-policy training improves sample efficiency, it poses a challenge that is unique to HRL. Since the worker's policy keeps changing under the manager's command, the same action produced by the manager in the past is highly probable to yield a different low-level behavior in the future. Therefore, past experiences collected in the replay buffer are not valid to train the manager any more. HIRO addresses this issue by *re-labeling* the action g_t in past transition $(s_t, g_t, \sum R_{t:t+c-1}, s_{t+c})$ to $\tilde{g}_t$, where $\tilde{g}_t$ is a different goal action chosen to maximize the probability of the past lower-level actions $a_{t:t+c-1}$ under the worker's current policy. That is,

$$\max_{\tilde{g}_{t:t+c-1}} \mu^{lo}\left(a_{t:t+c-1} | s_{t:t+c-1}, \tilde{g}_{t:t+c-1}\right) \tag{36}$$

In other words, we would like to know which goals would allow this new controller to repeat its old actions.

For comparison, HIRO is reported to outperform a FuN modification [35] on Ant Maze, Ant Fall, and Ant Push.

4.4.2 Decision states as subgoals

Like STRAW [27], InfoBot [44] also explores the concept of decision states where obvious behavior change is required. InfoBot defines the **decision states** in the scenario where the agent learns goal-conditioned policies in multi-goal environments. In these states, the agent learns to deviate from its habits, i.e. default policy, and make a right decision towards specific goal states. In other words, the **default policy** is what the agent follows in the absence of any knowledge about the goal, such as goal locations, directions, the relative distance to the goal. It can thus be formulated as the result of marginallizing out goal g in multi-goal policies $\pi_\theta(A|S, g)$:

$$\pi_0(A|S) = \sum_g p(g)\pi_\theta(A|S, g) \tag{37}$$

To recognize decision states, the agent is trained with an information regularizer:

$$I(A; G|S) = \mathbb{E}_{\pi_\theta}\left[D_{\mathrm{KL}}\left[\pi_\theta(A|S, G)|\pi_0(A|S)\right]\right] \tag{38}$$

Minimizing this regularizer encourages the agent to rely on its goal-independent habits as much as possible, and only deviates in decision states. Thus, this regularizer plays the role of *information bottleneck* which limits the amount of goal information used by the agent's policy during training.

In this respect, decision states are natural subgoals since they identify the boundaries between achieving different goals. Actively seeking out these subgoals can benefit further exploration. Note that decision states are similar in spirit to the notion of *bottleneck states* [48], which is defined as the intersections of a variety of rewarding trajectories. However, partitioning the state space purely using graph theory without considering the reward structure [49] will lead to a trivial explanation of bottleneck states, e.g., in a setup of a T maze. In contrast, decision states are more parsimonious and accurate.

The experiments have shown that training an agent with a goal bottleneck alone leads to more efficient policy transfer than a vanilla goal-conditioned A2C agent. For example, in MultiRoom navigation environments, the agent can discover a wall following strategy quickly when it is transferred to solve a larger task.

4.4.3 Directional subgoals

FeUdal Networks (FuN) [35] embodies a two-level Feudal framework by end-to-end differentiable neural networks, and generalizes the definition of subgoals from absolute locations to directions in latent state space. Using directions to direct the worker's policy other than locations provides the following advantages:

- Directional subgoal makes a clear connection to the **worker's transitions**, and constrains them to follow a specific distribution around that direction, thus is much **reliable** to achieve than reaching an arbitrary location.
- A directional subgoal assumes one degree of invariance, thus allows for **structural generalization** of corresponding sub-policies.
- The worker's pursuit of subgoals leads to a **semantically decomposed** latent space, where diverse directions are translated into meaningful behavioural primitives.

In order to allow better long-term credit assignment and tractable memorization, both manager and worker are modeled as LSTMs, which keep compressing latent states into their memories. Then given observation x_t and external reward R_t received from the environment, the architecture of FuN (Fig. 4) can be described from two views:

- **Manager**: At every c time steps, the manager outputs a unit vector g_t as the goal. However, treating g_t as a latent variable and training it by gradients coming from the worker would deprive g_t from any semantic meaning. Instead, it is desired that g_t is along advantageous directions promising to maximize future rewards:

$$\nabla g_t = A_t^M \nabla_\theta d_{\cos}\left(s_{t+c} - s_t, g_t(\theta)\right) \tag{39}$$

  where A_t^M is the manager's advantage function defined as $A_t^M = R_t - V_t^M\left(x_t;\theta\right)$. Cosine similarity $d_{\cos}(\cdot)$ measures the discrepancy between current advancing direction and the goal direction.

  Once a goal g_t is chosen, the worker's behavior can be predicted by the manager according to transition distribution $\pi_t^{TP} \triangleq p(s_{t+c}|s_t, g_t)$ if the worker indeed learned to follow the goal. Furthermore, if π_t^{TP} follows von Mises-Fisher distribution, then it can be updated by:

$$\nabla_\theta \pi_t^{TP} = \mathbb{E}\left[\left(R_t - V\left(s_t\right)\right) \nabla_\theta \log \pi_t^{TP}(\theta)\right] \tag{40}$$

  which is exactly equivalent to Eq. (39).

- **Worker**: The worker LSTM outputs an embedding vector for each possible action. Then the action embedding matrix is mixed with a goal embedding pooling over goals of past c time steps to produce a policy action. The worker is trained on a weighted sum of external rewards and intrinsic rewards $R_t + \alpha R_t^I$, where R_t^I is derived to encourage the worker to align with the goal direction:

$$R_t^I = 1/c \sum_{i=1}^{c} d_{\cos}\left(s_t - s_{t-i}, g_{t-i}\right) \tag{41}$$

FuN is trained using A3C and only applicable to tasks defined on discrete action space. Experiments are performed on Atari games such as sea quest and Montezuma's revenge to validate that FuN is capable of learning interpretable subgoals and sub-polices consistent with human experiences.

4.4.4 Actionable representation of subgoals

To find subgoals in state space, [43] proposes to distinguish two goal states functionally. That is, if two states require different actions to reach, then they are different. To capture factors of variation important for decision making from a state, (i.e. actionable representation), a distance metric is defined as:

$$D_{\text{Act}}\left(s_1, s_2\right) = \mathbb{E}_s\left[D_{KL}\left(\pi\left(a|s, s_1\right) \| \pi\left(a|s, s_2\right)\right) + D_{KL}\left(\pi\left(a|s, s_2\right) \| \pi\left(a|s, s_1\right)\right)\right] \tag{42}$$

Note that to make it hold for arbitrary initial state, the expectation is computed over all initial states s. And we assume that a policy π has already been trained with maximum entropy.

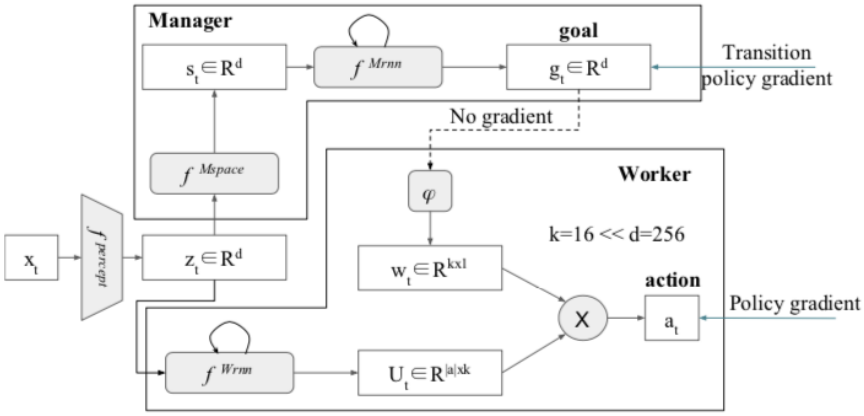

Figure 4: Architecture of FuN

To learn this actionable representation $\phi(s)$, one could optimize:

$$\min_{\phi} \mathbb{E}_{s_1,s_2} \left[\|\phi(s_1) - \phi(s_2)\|_2 - D_{\text{Act}}(s_1, s_2) \right]^2 \tag{43}$$

This actionable representation coarsely captures the dynamics of the environment, and also provides a good metric in reachability which can be used as intrinsic rewards in Feudal HRL.

4.4.5 Subgoals as entities-relations

In the previous sections, we have discussed a variety of subgoal representations in continuous control scenarios. However, all of them are studying state abstractions in a simplified, low-dimensional space, whose complexity is far from what real world presents in terms of objects and their relations. This gap prevents us from investigating the agent-world interaction in a right way. That is, *a right state representation should allow the agent to realize that state transitions of our physical world are a result of changes happened on object states (observations).* Therefore, it is reasonable to hypothesize that an appropriate state abstraction should be *structured or sufficiently compositional* [50]. For example, a state could be represented by a scene graph, and the state change could be computed by a Graph Neural Network (GNN). Then the question is how can we learn such state abstractions that support efficient decision making?

A recent work h-DQN [50] has taken first steps towards answering this question by *defining subgoals in the space of entities and relations*. The architecture of h-DQN is a general framing of two-layer Feudal framework (Fig. 5) which is composed of a meta-controller, a controller, and a critic. Both of the meta-controller and controller are trained by DQN but on extrinsic rewards and intrinsic rewards respectively. The Q functions for meta-controller and controller are denoted by $Q_2^*(s, g)$ and $Q_1^*(s, a; g)$. The critic receives subgoal g_t and gives intrinsic rewards $r_t(g_t)$ to the controller based on whether g_t has been reached, where r_t is predefined and task specific. Note that the probability ϵ in ϵ-greedy also depends on current empirical success rate of reaching g_t.

The h-DQN agent is evaluated on Montezuma's Revenge where the subgoals are defined in space of tuples $< entity_1, relation, entity_2 >$. The agent receives a reward when the tuple is evaluated as true. In this case, the agent gets score when it reaches a target location, e.g, key or door.

5 Subgoals and curricula generation

Curriculum learning [51] is a learning strategy where tasks are presented in an order with increasing difficulty for better generalization and faster learning. This progression of tasks can be described by sub-goals as way stations to the overall goal. Making a connection between subgoal and curriculum learning enables us to expand the scope of subgoal from space to time. Different from being in the hierarchical control scenario, a set of subgoals here not only indicates a decomposition of the target

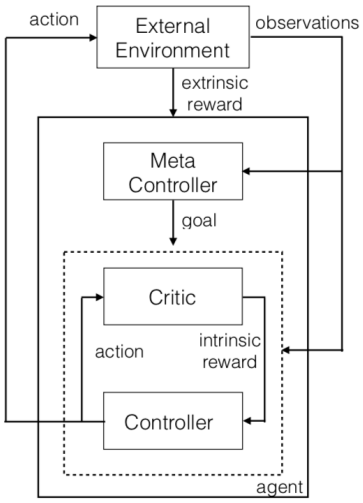

Figure 5: Architecture of h-DQN

task, but also presents a clear relation among its elements, i.e., the subgoals are arranged in an order of increasing learning difficulty.

In this section, we will introduce several methods that can generate the curricula automatically by a set of self imposed subgoals.

5.1 Emergence of curricula from self-play

Unlike previous Fedual hierarchical models, Self-Play model [52] pre-trains the low level controller, i.e. the worker, by unsupervised asymmetric self-play before gets it directed by the manager via subgoals. During self-play, the agent keeps inventing new tasks with increasing difficulty and complexity for itself to solve via the goal embedding. The diversity of goals is ensured by an adversarial reward structure.

Specifically, in self-play phase, consider Alice and Bob as two separate "minds" of the agent and they takes turns in control. In each round of game, Alice first executes her policy in the environment for T_A time steps and ends up at state s^*. Then Bob takes actions for T_B time steps, and succeeds if at any time step he is close to s^*. Either Alice or Bob is rewarded 1 and the other 0. This reward structure encourages Bob to master a task quickly and forces Alice to set new unexplored task to challenge Bob. The run and chase game between Alice and Bob automatically generates a curriculum of exploration. Imposing a similar time limit T_A and T_B ensures that the environment is always in the right difficulty for Bob to improve. To strengthen the role of Bob as a learner and Alice as a explorer, Bob is trained to imitate ground truth actions from Alice. And Alice is encouraged to devise diverse tasks by maximum entropy.

A continuous low dimensional goal embedding g_t is learned as part of Bob's policy:

$$g_t = E\left(s^*, s_t^B\right) \tag{44}$$

where g_t could be a relative measure $\phi\left(s^*\right) - \phi\left(s_t^B\right)$ or an absolute representation of the target $\phi\left(s^*\right)$. When the tasks involve manipulations of the environment, self-play learning distills the controllable parts of the environment into the goal embedding, and ignores those the agent cannot change, for instance the static background or random elements.

Next, in the hierarchical control phase, Alice quits and Charlie joins as a high level controller. Charlie generates goal g_t leveraging the learned goal embedding space to guide Bob to complete the target task. After self-play, it is easier for Charlie to specify achievable subgoals that make up complex tasks (Fig. 6).

The model has been evaluated on two tasks. In the control task, an Ant is required to gather randomly placed objects. It is shown that Alice is able to propose diverse goal positions in all directions. The

hierarchical controller enables the agent to make long-distance exploration which is difficult for an agent purely trained on flat atomic actions. The second game KeyDoor asks the agent to find a random located treasure by first picking up a key and then opening the door. Alice is observed to devise an increasingly complex curriculum throughout self-play. The initial tasks Alice proposed are basic and just involve movement. Gradually, the agent learned to unlock the door and navigate to another room. And finally the space of tasks afforded by the environment has been explored. Moreover, the goal space is shown to reflect the controllable aspects of the environment.

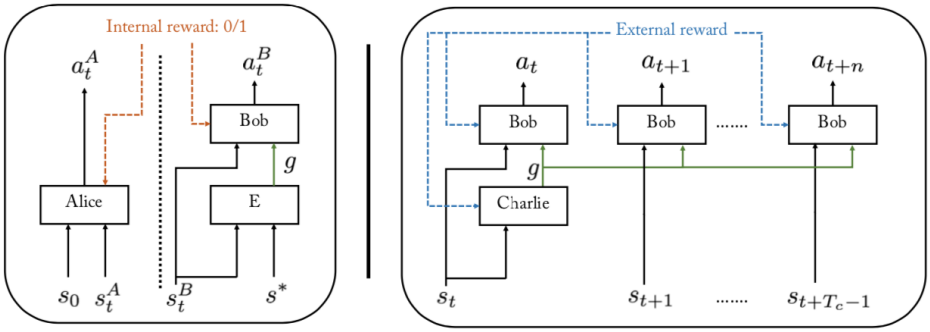

Figure 6: Architecture of Self-Play model

5.2 Other curricula generation methods

Driven by the challenge posed in many robotic tasks that an agent is only given sparse or even binary rewards, Hindsight Experience Replay (HER) [16] proposes to extract rewards from unsuccessful experiences. An implicit curriculum is built by labeling previously reached states as goals other than the one the agent was trying to achieve when replaying each episode. Those goals for replay shift naturally from the ones easy to achieve even by a random policy to more difficult ones. Unlike explicit curriculum, HER does not require any control on the distribution of initial environment states. The simplest strategy to sample goals for replay is to use the final state achieved in each episode.

Similar to HER, method [45] also considers the problem of goal generation. However, instead of sampling goals along past trajectories, this approach tries to gradually generate goals matching the agent's current capability.

Let each goal $g \in \mathcal{G}$ correspond to a set of states S^g. The agent is rewarded according to function r^g if it is in any state $s \in S^g$. A curriculum can be reliably built only when the goals it is made of can be consistently reached by some policy. The overall objective of an agent is to find a policy to achieve a high reward for a set of goals sampled from p_g:

$$\pi^* \left(a_t | s_t, g\right) = \arg\max_{\pi} \mathbb{E}_{g \sim p_g(\cdot)} R^g(\pi) \tag{45}$$

To optimize the coverage of goals, the agent is advised to always learn from a set of goals with suitable difficulty. That is, the goals are reachable for the agent and also provide enough improvement space:

$$GOID_i := \{g : R_{\min} \leq R^g\left(\pi_i\right) \leq R_{\max}\} \subseteq \mathcal{G} \tag{46}$$

where $R_{\max}$ is an upper bound of the performance above which the agent prefer to concentrate on new goals. And $R_{\min}$ requires the agent to receive enough reward signal for learning. To satisfy this restriction, the sampling might by performed repeatedly from a small set of already mastered goals.

To provide a continuous goal-space representation such that a goal-conditioned policy can efficiently generalize over the goals nearby, a generative adversarial network (GAN) is trained to generate goals satisfying $GOID_i$.

By combining goal relabeling technique from [16] and goal generation, paper [53] learns a latent space of high dimensional observations by β-VAE, which enables the agent to imagine goals and practice to achieve them.

POET [54] goes beyond curriculum learning by inventing its own diverse and expanding curricula not in a strict sequence, but asynchronously and in parallel like a tree. This diverse multi-path discovery progression is accomplished by pairing the environment generation and agent optimization in the spirit of combinatorial multi-objective evolutionary algorithm (CMOEA) and a minimal criterion coevolution (MCC). The discovery of stepping stones or subgoals is based on novelty search where the individuals with behaviors most different from their ancestor are selected. These subgoals could be transferred between environments to catalyze innovation. Each active environment is encoded as a vector which could be modified to a new one by random mutation.

In the experiments, a bipedal walker equipped with a LIDAR is required to navigate through various terrains with diverse obstacles. The environment is chosen to be a testbed is because it is easy to modify and fast to simulate. Experimental results have shown that POET is able to find subgoals leading to solutions to very challenging environments, which could neither be solved by direct optimization nor building a single path curriculum.

6 Conclusions and future directions

In this survey, we study the problem of finding right structures in deep reinforcement learning. We investigate hierarchical reinforcement learning techniques by looking into two frameworks, Options framework and Feudal framework, which focus on abstractions of policy space and state space respectively. Further, we study the concept of subgoal first in the context of HRL, then move on to explore its connections to curriculum learning.

Although modern RL algorithms have extensively explored the problem of structure discovery for temporal extended behaviors and goal space, the structure inside a state has neither received enough attention nor been exploited sufficiently in solving problems with rich semantic information [55] or requiring complex planning [56]. As an initial attempt, we would like to study this open direction in simplified scenarios, such as block stacking [57] and Montezuma's Revenge. And another interesting and challenging problem we would like to delve into is how to automatically generate subgoals by observing the consequences of environment manipulation.

Acknowledgments

The author would like to thank the research exam committee members, Professor Shachar Lovett, Professor Taylor Berg-Kirkpatrick, and Professor Lawrence Saul. The author would also like to thank Professor Hao Su for his insightful comments.

References

[1] Richard Swinburne. *The Nature of Explanation*, pages 23–51. 03 2004.

[2] Charles Kemp and Joshua B Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences of the United States of America*, 105:10687–92, 09 2008.

[3] Noam Chomsky. Three learnable models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 12 1956.

[4] Matthew Botvinick. Hierarchical models of behavior and prefrontal function. *Trends in cognitive sciences*, 12:201–8, 06 2008.

[5] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14:357–64, 08 2010.

[6] Elizabeth S. Spelke, Karen Breinlinger, Janet Macomber, and Kristen Jacobson. Origins of knowledge. *Psychological review*, 99:605–32, 11 1992.

[7] Elizabeth S. Spelke. Core knowledge. *The American psychologist*, 55:1233–43, 12 2000.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[10] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, and Tara Sainath. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97, November 2012.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations(ICLR)*, 2015.

[12] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *International Conference on Learning Representations (ICLR)*, 2015.

[13] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.

[14] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations, pages 77–109. MIT Press, Cambridge, MA, USA, 1986.

[15] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[16] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5048–5058. 2017.

[17] Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv:1904.05538*, 2019.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, L Robert Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Fong Celine Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[20] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, pages 3215–3222. AAAI Press, 2018.

[21] Xingchao Liu, Tongzhou Mu, and Hao Su. Transfer value or policy? a value-centric framework towards transferrable continuous reinforcement learning. In *Workshop on Deep RL at NIPS*, 2018.

[22] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[23] Andrew G. Barto, George Konidaris, and Christopher Vigorito. *Behavioral Hierarchy: Exploration and Representation*, pages 13–46. 09 2013.

[24] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61, 01 2018.

[25] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pages 166–175, 2017.

[26] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. META LEARNING SHARED HIERARCHIES. In *International Conference on Learning Representations (ICLR)*, 2018.

[27] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, and koray kavukcuoglu. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3486–3494, 2016.

[28] Richard Bellman. *Dynamic programming*. Princeton University, 1957.

[29] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS)*, pages 1057–1063, 1999.

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[31] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[32] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[33] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[34] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, 2016.

[35] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3540–3549, 2017.

[36] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[37] John D. Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2018.

[38] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 12 2014.

[39] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 271–278. 1993.

[40] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pages 1312–1320, 2015.

[41] Max Jaderberg, Volodymyr Mnih, Wojciech Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations (ICLR)*, 2017.

[42] Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3303–3313. 2018.

[43] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *NeurIPS Deep RL Workshop 2018*, 2018.

[44] Anirudh Goyal, Riashat Islam, DJ Strouse, Zafarali Ahmed, Hugo Larochelle, Matthew Botvinick, Sergey Levine, and Yoshua Bengio. Transfer and exploration via the information bottleneck. In *International Conference on Learning Representations (ICLR)*, 2019.

[45] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1515–1528, 2018.

[46] David Silver. Deep reinforcement learning with subgoals. *Hierarchical Reinforcement Learning Workshop at NIPS*, 2017.

[47] N. Dilokthanakul, C. Kaplanis, N. Pawlowski, and M. Shanahan. Feature control as intrinsic motivation for hierarchical reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[48] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 361–368. Morgan Kaufmann Publishers Inc., 2001.

[49] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 295–306. Springer-Verlag, 2002.

[50] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3675–3683. 2016.

[51] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 41–48, 2009.

[52] Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv:1811.09083*, 2018.

[53] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NIPS)*, pages 9209–9220, 2018.

[54] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv:1901.01753*, 2019.

[55] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart J. Russell, Georgia Gkioxari, and Yuandong Tian. Learning and planning with a semantic model. *aXiv:1809.10842*, 2018.

[56] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations (ICLR)*, 2019.

[57] Victor Bapst, Alvaro Sanchez-Gonzalez, Carl Doersch, Kimberly L. Stachenfeld, Pushmeet Kohli, Peter W. Battaglia, and Jessica B. Hamrick. Structured agents for physical construction. *International Conference on Machine Learning (ICML)*, 2019.