# 10-708 Final Report: Investigating Max-Entropy Latent-Space Policies for Hierarchical Reinforcement Learning

**Siddharth Ancha (sancha)** [1] **Mengxiong Liu (mengxiol)** [1] **Xingyu Lin (xlin3)** [1]

## Abstract

In this paper, we investigate previous work Haarnoja et al. (2018a) that combines maximum entropy (Max-Ent) reinforcement learning and hierarchical reinforcement learning (HRL). This formulates HRL as inference over probabilistic graphical models with a particular latent variable structure. Such formulation enables training of multiple policies in a stacked, hierarchical fashion. We hypothesize that the reason that a stacked policy outperforms the low-level policy is that the low-level policy provides a good behaviour policy which eases exploration. We provide theoretical analysis for the effect of the behaviour policy in tabular cases and propose a new baseline which uses a low-level policy purely as the behaviour policy for training another policy. We find that our baseline outperforms the latent space. We additionally provide and analyze visualizations of the learned low-level, latent space policy, and we find a surprising lack of diversity in low-level skills.

## 1. Introduction

In this paper, we aim to investigate a hierarchical reinforcement learning framework built on a probabilistic graphical model formulation of maximum entropy reinforcement learning. Maximum Entropy RL (Todorov, 2007; Ziebart et al., 2008; Haarnoja et al., 2018c; Levine, 2018) frames reinforcement learning as inference over a particular graphical model (Haarnoja et al., 2018c; Levine, 2018). The states and actions of the Markov Decision Process (MDP) act as nodes of the model, while the MDP transition function defines the edges. It can be shown that optimizing the standard reinforcement learning objective, with an additional entropy term, can be formulated as variation inference over this model. Max-Ent RL has shown to be useful for robustness, exploration, diversity and compositionality of behaviors.

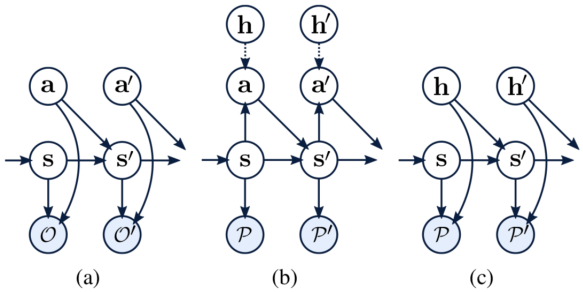



*Figure 1.* From (Haarnoja et al., 2018a). (a) regular graphical model for max-ent RL learning, (b) learning a lower-level policy by reparametrizing $\pi(a_t \mid s_t)$ in terms of $h_t$ with a prior, (c) viewing reparametrization as original problem with modified dynamics for higher-level policy learning.

Haarnoja et al. (2018a) extend this graphical model, using latent variables, as a framework for hierarchical reinforcement learning (HRL). Multiple layers of the graphical model, each representing a sub-policy, can be stacked on top of each other, and connected via latent variables. This resembles a Deep Belief Network (Hinton & Salakhutdinov, 2006). Much like a DBN, Haarnoja et al. (2018a) train the model layer-wise. Layers that are trained first represent lower-level sub-policies. These lower-level policies induces behaviors which can be used by higher-level sub-policies to more easily learn complex tasks.

Specifically, Haarnoja et al. (2018a) re-parametrize the action space of the lower-level policy function in terms of latent variables $h_t$, that are sampled from uniform priors (see fig. 1(b)). This model can then be viewed as similar to the original graphical model for regular Max-Ent RL (shown in fig. 1(a)), where latent variables $h_t$ act as actions to be learned by a higher-level policy (see fig. 1(c)). Importantly, the dynamics of fig. 1(c) has been modified as the lower-level policy is now 'embedded' into the dynam-

[1]Carnegie Mellon University, Pittsburgh, PA 15213, USA.

*10708 Class Project,* Spring, 2019.

ics. This means that the dynamics has been biased to follow behaviors produced by the lower-level policy. Usually lower-level policies are trained on auxiliary rewards that are correlated with, or helpful to learning the true reward function by a higher-level policy.

We ask the question: Why is a hierarchical policy stacked in such a way performs better than a single-layer policy? We hypothesize that the reason that a stacked policy outperforms the low-level policy is that the low-level policy provides a good behaviour policy which eases exploration. This begs the following question: if one has already trained a lower-level policy, then, compared to using that for reparametrization or modified dynamics, would learning a new policy with the pre-trained low-level policy as the behaviour policy yield a comparable or even better policy?

In this work, we provide theoretical analysis for the effect of the behaviour policy in tabular cases and compare our proposed baseline with the hierarchical latent space policy (Haarnoja et al., 2018a). We find that our baseline outperforms the latent space. We additionally provide and analyze visualizations of the learned low-level, latent space policy, and we find a surprising lack of diversity in low-level skills.

## 2. Background & Related Work

### 2.1. Maximum Entropy Reinforcement Learning

Previous works (Todorov, 2007; Ziebart et al., 2008; Levine, 2018) have shown that optimal control can be framed as a probabilistic inference problem. This leads to Maximum Entropy RL, where the agent's goal is to jointly maximize the expected return and entropy of the policy. Maximum Entropy RL provides several benefits, including but not limited to structured exploration (Haarnoja et al., 2018c; 2017), learning diverse skills (Eysenbach et al., 2018) and skill composition (Haarnoja et al., 2018b) for high dimensional continuous control tasks. Under the Maximum Entropy RL framework, the probability of choosing an action is proportional to the exponential of Q-function, i.e. the optimal policy follows a Boltzmann distribution, which could potentially be highly multi-modal.

Haarnoja et al. (2017) proposed Soft Q-learning (SQL) based on Soft Q-iteration. It performs temporal difference to learn the Q-function of the optimal maximum entropy policy. However, given that the action space is continuous and the Q-function is highly multi-modal, it is not reasonable to approximate the optimal policy by distributions with limited expressiveness, like the Gaussian distribution. Hence, SQL employs an actor parameterized by a neural network to sample from the optimal policy using Stein Variational Gradient Descent (Liu & Wang, 2016). Haarnoja et al. (2018b) adopts SQL (Haarnoja et al., 2017) to compose multiple policies for different tasks by summing all the Q-functions and uses a sampler actor to draw samples from the resulting Boltzmann distribution. Haarnoja et al. (2018b) also provide a sub-optimality bound of the composed policy compared to the policy that directly maximizes the summed reward. Haarnoja et al. (2018c) propose soft actor-critic based on soft policy iteration that results in a DDPG (Lillicrap et al., 2015) like update rule with additional policy entropy term in the target Q-function and target value function. In contrast to the sampling technique used in SQL, SAC parameterizes a Gaussian policy and minimizes the KL-divergence between the Gaussian policy and the optimal policy. SAC is observed to exhibit efficient exploration and improved convergence on several benchmark tasks. Eysenbach et al. (2018) adopt SAC to perform unsupervised skill discovery by maximizing an information theoretic objective and observes emergence of diverse skills even without any reward function.

### 2.2. Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning is usually based on the option framework (Sutton et al., 1999). In this framework, an option consists of a lower-level policy and a state dependent termination function which decides when to terminate the lower-level policy and hand the control over to the higher-level policy. The latter provides the next option to execute.

While the high-level policy can be learned by maximizing the external reward in a semi-MDP, it remains an open problem on how to obtain the option, or the lower-level policy. Some previous works manually design the lower-level policy, e.g. motion primitives in the context of robotic locomotion (Heess et al., 2016). This approach, while effective, excessively relies on domain specific knowledge and lacks flexibility. Another approach is to learn both the high-level policy and the low-level policy in an end-to-end manner, such as the option-critic framework (Bacon et al., 2017), which learns the lower-level policy and the termination function based on the advantage function in a continuous way. However, often times, the end-to-end trained policy degenerates into a flat policy as the higher-level policy keeps using only one option and thus loses the benefit of using a hierarchical policy structure.

Many heuristics have also been proposed to learn the option, such as maximizing the mutual information between the option and the state distribution (Florensa et al., 2017), bounding the mutual information between the actions and top-level actions (Daniel et al., 2012), and using entropy regularization to ensure diversity in lower-level options

(Bacon et al., 2017).

Haarnoja et al. (2018a) similarly adopt Max Entropy RL to provide diversity in the lower-level policy. However, a notable difference between Haarnoja et al. (2018a) and the option framework is that, Haarnoja et al. (2018a) does not have the usual temporal abstraction seen in the option framework. Initially, the 'lower-level' policy is learned by directly optimizing the external reward and augmented rewards, with latent random variables as additional input. Then, layers of a 'higher-level' policy can be stacked on top of the base policy that control the latent variables. There does not exist a higher level policy on the semi-MDP which only makes a decision after a number of time steps. This separation from the previous frameworks lead to our motivation for providing a better understanding of compositionality or hierarchy in this paper. For example, is there semantic meaning for the learned latent variables and the learned base policy. What benefit does the hierarchical policy bring to some MuJoCo tasks which do not exhibit a natural compositionality?

### 2.3. Control as Inference

(Levine, 2018) cast control problem to posterior inference on graphical model. The graphical model is composed of state action pairs at every time step $s_t, a_t$ and an associated optimality variable $\mathcal{O}_t$. The transition dynamics $p(s_{t+1}|s_t, a_t)$ and the optimality condition $p(\mathcal{O}_t|s_t, a_t)$ are represented as edges where $p(\mathcal{O}_t = 1|s_t, a_t) \propto e^{r(s_t,a_t)}$. The probability of optimal trajectory conditioned on the optimality variable defined as the following

$$p(\tau|\mathcal{O}_{0:T}) \propto p(s_0) \prod_{t=0}^{T} p(a_t)p(s_{t+1}|s_t, a_t) \exp(r(s_t, a_t)) \tag{1}$$

From the above expression, we can infer the optimal action at a given state as

$$p(a_t|s_t, \mathcal{O}_{t:T}) \propto \frac{p(\mathcal{O}_{t:T}|s_t, a_t)}{p(O_{t:T}|s_t)} = \frac{\beta_t(s_t, a_t)}{\beta_t(s_t)} \tag{2}$$

where

$$\beta_t(s_t, a_t) = \int_{\mathcal{S}} \beta_{t+1}(s_{t+1})p(s_{t+1}|s_t, a_t)p(\mathcal{O}_t|s_t, a_t)ds_{t+1} \tag{3}$$

However, directly inferring optimal action from the above expression is problematic. First, in practice the action lies in a high dimensional continuous space and may not have a well defined parametric form, which makes it intractable to sample from. Second the above expression assumes that the transition dynamics can also be optimized to favor the optimal actions, which in practice is not true. To satisfy the constraint that the optimal action cannot modify transition dynamics and to make sure the policy follows a tractable parametric form, define the variational distribution of the optimal trajectory as the following

$$q(\tau) = p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t) \tag{4}$$

Then we arrive at the following evidence lower bound

$$\log p(\mathcal{O}_{0:T}) \geq -\mathrm{D}_{\mathrm{KL}}(q(\tau)\|p(O_{0:T}, \tau)) \tag{5}$$

With a uniform action prior, the KL divergence can be reduced to the maximum entropy RL objective

$$E_{\tau\sim\rho_\pi(\tau)}[\sum_{t=0}^{T} r(s_t, a_t) + \mathcal{H}(\pi(\cdot|s_t)) \tag{6}$$

This objective is can be optimized using Soft Q-learning (Haarnoja et al., 2017) or SAC (Haarnoja et al., 2018c).

## 3. Methods

### 3.1. Latent Variable Policy for Hierarchical Reinforcement Learning

(Haarnoja et al., 2018a) extend the standard graphical model 1a by introducing a hidden variable $h_t$ and condition the action $a_t$ on $h_t$ and $s_t$ at each state $s_t$. At each time step, $h_t$ is first sampled from its prior $p(h_t)$, then sample an action from $\pi(a_t|s_t, h_t)$. Therefore, a new set of optimality variable $\mathcal{P}_t$ can be introduced for the hidden states and $\mathcal{P}_t$ may or may not be the same as $\mathcal{O}_t$ depending on the tasks. This induces a new graphical model 1b, which naturally can be considered as a hierarchical RL framework. The high level policy is formulated as the posterior distribution of $h_t$ given optimality variable $\mathcal{P}_t$, similar to the traditional setting 1a where the policy is formulated as approximating the posterior distribution of $a_t$ given optimality variable $\mathcal{O}_t$. Then the low-level policy is formulated as approximating the distribution of $a_t$ given $s_t$ and $h_t$. More layers of hidden variables can be added in this graphical model to form a multi-level hierarchical RL framework. Note that integrating $a_t$ out from 1b we arrive at a reduced graphical model 1c, which resembles the graphical model in 1a. Note that $h_t$ in 1c is equivalent to $a_t$ in 1a and $\mathcal{P}_t$ in 1c is equivalent to $\mathcal{O}_t$ in 1a. Therefore, 1c can be formulated as a new MDP with $h_t$ as actions and $a_t$ embedded into the environment. The new transition dynamics then becomes

$$p(s_{t+1}|s_t, h_t) = \int_{\mathcal{A}} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t, h_t)da_t \tag{7}$$

To mitigate the intractability of integrating over $a_t$, the high level policy $\pi(a_t|s_t, h_t)$ is parameterized as a deterministic function in practice give $s_t$ and $h_t$. With the low-level

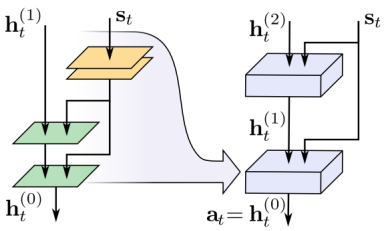



*Figure 2.* LSP Policy Network (Haarnoja et al., 2018a). The yellow layers are fully connected layers that encode the state $s_t$. The green layers are coupling layers used in Real NVP.

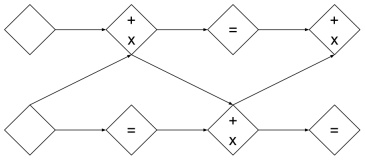



*Figure 3.* Real NVP (Dinh et al., 2016)

policy embedded into the environment, the new MDP can potentially become an easier problem.

### 3.2. Policy Architecture

The policy is parameterized as a deterministic function $a_t = f(s_t, h_t)$. Suppose $f$ is a bijective function with respect to $a_t, h_t$ but not necessarily $s_t$, then the density of $a_t$ can be expressed using a change of variable formula

$$\pi(a_t|s_t) = p(h_t)|\det(\frac{df(h_t; s_t)}{dh_t})|^{-1} \tag{8}$$

To ensure the transformation from $h_t$ to $a_t$ is invertible and Jacobian is tractable, Haarnoja et al. (2018a) adopted Real NVP Dinh et al. (2016) architecture 3. The policy is composed of a sequence of coupling layers. Each of the coupling layers applies a invertible transformation by scaling and adding an offset to some dimensions of the input vector conditioned on the other dimensions of the input vector while keeping the other dimensions intact. Let $x$ be the input vector of the coupling layer of dimension $D$, $y$ be the output of coupling layer also of dimension $D$, $d$ be the partition dimension. The coupling layer performs the following transformation

$$\begin{cases} y_{1:d} & = x_{1:d} \\ y_{d+1:D} & = x_{d+1:D} \odot \exp g(x_{1:d})) + t(x_{1:d}) \end{cases} \tag{9}$$

where $g$ and $t$ can be arbitrary complex functions and not necessarily invertible. The inverse transformation is then

$$\begin{cases} x_{1:d} & = y_{1:d} \\ x_{d+1:D} & = (y_{d+1:D} - t(y_{1:d})) \odot \exp(-s(y_{1:d})) \end{cases} \tag{10}$$

And the jacobian of the forward transformation is

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{d+1.D}}{\partial x_{1:d}^T} & \text{diag}(\exp[s(x_{1:d})]) \end{bmatrix} \tag{11}$$

Note that after one coupling layer of transformation, half of the input vectors are kept intact. To mitigate this problem, Dinh et al. (2016) performs a switch of grouping after each coupling layer, i.e. transforming the set of dimensions that are kept intact in the previous layer, to make sure all hidden units have gone through some nonlinear transformation, as illustrated in 3. The hidden state $h_t$ is transformed to action $a_t$ through a sequence of coupling layers. To ensure the action also depend on the state $s_t$ and to guarantee the expressiveness of the policy network, Haarnoja et al. (2018a) also condition the function $g$ and $t$ on state $s_t$. The forward transformation then becomes

$$\begin{cases} x_{1:d} & = y_{1:d} \\ x_{d+1:D} & = (y_{d+1:D} - t(y_{1:d}, s_t)) \odot \exp(-s(y_{1:d}, s_t)) \end{cases} \tag{12}$$

as illustrated in 2.

### 3.3. Full Algorithm

We start by training the lowest level policy directly on the original MDP by optimizing the maximum entropy objective stated above using SAC Haarnoja et al. (2018c). Then we freeze the low-level policy and embed it into the environment. After that the hidden states become the new actions. The new reward functions then becomes $r(s_t, f(s_t, h_t))$ and the transition dynamics is becomes as $p(s_{t+1}|s_t, h_t) = p(s_{t+1}|s_t, f(s_t, h_t))$. The higher level policy is then trained under the transformed MDP using SAC. We can stack more layers on top of the higher level policy to form a multi-level policy and train all levels in the same fashion described above.

### 3.4. Interpretations

We propose two possible interpretations of the benefits of stacking higher level policies on top of lower level policies. First, low-level policy transforms the environment such that learning from the transformed environment is easier for the high level policy. Consider the case where low-level policy is initialized to a random policy, then learning the higher level policy on the transformed environment is no easier

than learning the lower level policy. However, if the lower level policy is initialized to one that is close to the optimal policy, even a random higher level policy still manages to achieve high total return on the transformed environment. Due to the invertibility of the all levels of policies, the higher level policy can learn to correct mistakes learned by lower level policies. Another explanation is that the lower level policy provides a good behavior policy to the higher level policy. Suppose the transformation $a_t = f(s_t, h_t)$ is initialized to an identity mapping from $h_t$ to $a_t$, then executing the high level policy is equivalent to executing the low-level policy. Note that it is feasible to initialize the high level policy as an identity mapping since we can initialize the parameters of the $g$ and $t$ to be all close to 0. Then the low-level policy can be considered as a behavior policy to the high level policy. To study how much does each of two factors contribute to the benefits of LSP, we propose a comparison to disentangle the two factors. Specifically, we propose to train a LSP with one level of hierarchy from scratch with pre-trained behavior LSP also with one level of hiearchy. We hypothesize that the improved performance introduced by stacking extra layers of latent space policy might be merely because of a good initialization provided by the low-level policy. We'd like to see if we are able to achieve similar performance using a good initialization without introducing extra hierarchies. Given a pre-trained policy and its value and Q function, we'll train a new policy $\pi_\phi$, its value function $V_\psi$ and Q function $Q_\theta$, from scratch using pre-trained policy $\pi_{\bar{\phi}}$ as behavior policy. First, we'll use the behavior policy to collect trajectories to fill in the replay buffer and use the state action pairs sampled from the replay buffer to perform temporal difference update on $\pi_\phi, V_\psi, Q_\theta$. After burning in for $T$ iterators, we'll start filling in the replay buffer with trajectories collected by the new policy $\pi_\phi$ to continue updating $\pi_\phi, V_\psi, Q_\theta$. We'll follow the update rule in SAC (Haarnoja et al., 2018c). The trajectories collected by the behavior policy gives the new policy a good initialization while $T$ prevents the new policy $\pi_\phi$ from saturating to the behavior policy $\pi_{\bar{\phi}}$.

### 3.5. Analysis

How much does a good behaviour policy help with policy learning? Intuitively, a good behaviour policy guides the exploration towards states with high return. In this section, we attempt to analyze the return in the simplest tabular cases and show that, if the behaviour policy is already optimal, the learning of the value function will be exponentially fast with respect to the number of samples.

#### 3.5.1. Problem Definition

We start with a discrete-time finite-horizon Markov decision process (MDP), with a finite state set $\mathcal{S} = \{s_1, \ldots, s_N\}$ and a finite action set $\mathcal{A} = \{a_1, \ldots, a_M\}$. For simplicity, we assume that the environment has a deterministic transition dynamics $s_{t+1} = f(s_t, a_t)$, and a bounded reward function $|r(s_t, a_t)| \leq R$. Some other notations include: $\rho_0 : \mathcal{S} \to R_+$ is the distribution of the initial state $s_0$, $\gamma \in [0, 1]$ is the discount factor and $T$ is the length of a horizon. The value function is defined by the accumulated, discounted future return at state $s$, follow policy $\pi$: $V_\pi(s) = E_\pi[\Sigma_{i=t+1}^{\infty} \gamma^{i-t-1} R_i | S_t = s]$. There always exits a deterministic optimal policy, denoted as $\pi^*(s_t)$.

#### 3.5.2. Learning with optimal behaviour policy converges exponentially

In the tabular MDP case, following a certain policy, the probability of a trajectory $\tau = (s_0, a_0, s_1, \ldots, s_T)$ can be written as

$$p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

We can then calculate the marginal distribution for each state $p(s)$ as:

$$\begin{aligned} p(s) &= \sum_{t=0}^{T} p(s_t = s) \\ &= \sum_{t=0}^{T} \sum_{s_0} \sum_{a_0} \cdots \sum_{a_t - 1} \sum_{a_t + 1} \cdots \sum_{s_T} \\ &\qquad p(\tau = (s_0, a_0, \ldots, s_t = s, \ldots, s_T)) \end{aligned} \tag{13}$$

We present our finding in the following lemma:

**Lemma 1.** *Assume that the optimal policy $\pi^*$ is used as the behaviour policy and the tabular Q function $\hat{Q}(s, a)$ is initialized pessimistically, i.e. $\hat{Q}(s, a) \leq -TR$. When using Monte Carlo method to learn the Q function with $D$ samples, with a probability of at least*

$$1 - \sum_{i=1}^{N} \left[1 - p(s_i)\right]^D,$$

*where $s_i$ is the $i^{th}$ state, the policy derived from $\hat{Q}$ will be the same as the optimal policy.*

*Proof.* In the tabular case with deterministic transition dynamics and a deterministic optimal policy, we can see that we only need one trajectory in order to determine the Q values for all the state action pairs encountered on this trajectory, simply by recording the expected future return for

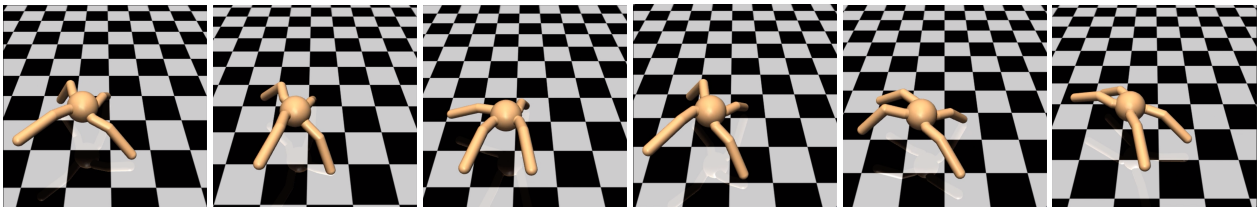


*Figure 4.* Lower-level policy learned for the multi-direction ant environment; the reward is proportional to the velocity obtained by the ant regardless of the direction of motion. We find that the ant learns to only move in one specific direction (bottom-left), for multiple policy rollouts.

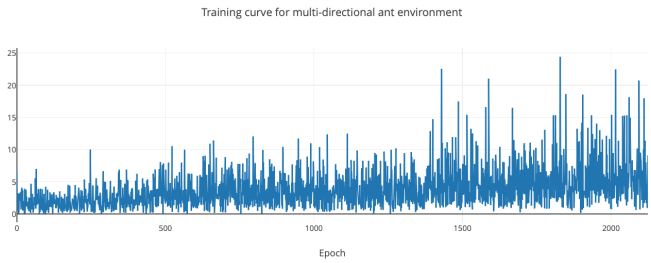
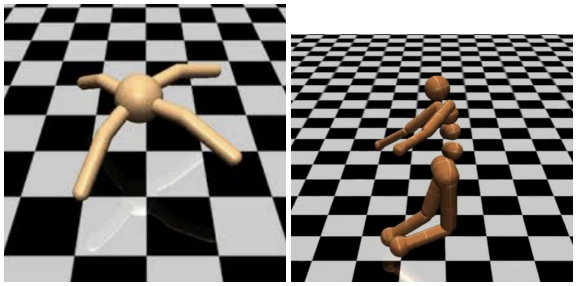



*Figure 5.* Training curve for ant in multi-directional setting where reward is given for attaining high velocities.



(a) Ant (rllab) environment. (b) Humanoid (rllab) environment.

*Figure 6.* Visualization of the mujoco environment.

each pair. Assume that a state $s$ is visited and $a^* = \pi^*(s)$. As $\pi^*$ is deterministic,

$$\hat{Q}(s, a^*) \geq -RT \geq \hat{Q}(s, a), \forall a \neq a^*.$$

Thus, we can see that once a state is visited by the behaviour policy, the learned Q function derived from this state will already yield the optimal action, i.e.

$$\text{argmax}_a \hat{Q}(s, a) = \pi^*(s).$$

To derive the full optimal policy from the learned tabular Q function, we simply need to cover all the states:

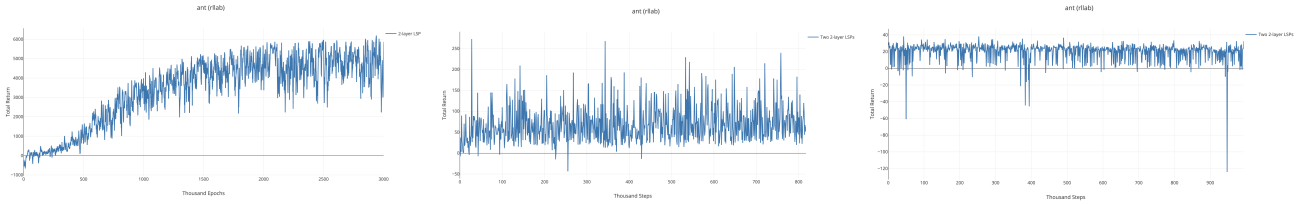
$$\begin{aligned} &P(\text{Visit all states with } D \text{ samples}) \\ &= 1 - P(\text{One state is not visited}) \\ &\geq 1 - \sum_{i=1}^{N} P(s_i \text{ is not visited}) \\ &= 1 - \sum_{i=1}^{N} \left[1 - p(s_i)\right]^D \end{aligned} \tag{14}$$

□

As each of the term $\left[1 - p(s_i)\right]^D$ goes to zero exponentially with respect to $D$, we can also approach the optimal policy with the learned Q function at an exponential rate. The reason we can achieve this is that the behaviour policy is already optimal and thus we do not need to do any exploration and only focus on learning Q function on state action pairs that achieve the highest return. While our analysis here does not apply to the real cases where the behaviour policy is not optimal, the MDP is continuous and has stochastic dynamics, we hope to emphasize the fact that having a good behaviour policy greatly reduces the complexity in exploration and thus motivates our proposed baseline for SAC with latent space policy.

## 4. Experiments

We modify the code from `https://github.com/haarnoja/sac`. We run our experiments on two Mujoco environments, ant (rllab) and humanoid (rllab), as shown in Figure 3. We first train a 2-layer latent space policy (LSP) for either 3 million steps or 6 million steps on humanoid. Then we freeze the low-level policy, add an additional level of 2-layer LSP and train for another 3 million steps for each of the trained lower level policy. A similar procedure is done for the ant environment, by first training a lower level LSP for 1.5 million steps and 3 million steps and then train for another 3 million steps with a stacked LSP on top of it. We observed that for both ant and humanoid, adding another level of policy does not introduce any observable improvement but we also did not observe any performance drop as observed in Figure 8.



(a) Training curve with a 2-layer latent space policy.

(b) Training curve with two 2-layer LSP, starting with 1.5M time steps

(c) Training curve with two 2-layer LSP, starting with 3M time steps

*Figure 7.* Training curves on Ant (rllab).

### 4.1. Multi-directional low-level Policy

Haarnoja et al. (2018a) perform an experiment where they train a low-level policy in the ant environment for a reward that is *not the same* as the final reward to be maximized. Basically, they train an agent in the ant environment to learn a diverse range of behaviors that would be useful and make it easier to learn other reward functions.

The setup of the experiment is as follows. The goal is for an ant that is bounded by a maze, to navigate towards certain goals. This requires the ant agent to both *locomote*, i.e. learn to control its actuators to move around. Additionally, it needs to direct this motion towards navigating to specified goals. Haarnoja et al. (2018a) train a two-level hierarchical policy to achieve this.

First, they train the ant, *in the absence of a grid*. The reward is proportional to the magnitude of the velocity achieved by the ant, *regardless of the direction*. This encourages the ant to move in a some direction instead of struggling to move while staying put.

Then, a higher-level latent space policy is trained to navigate the ant in a maze, using the low-level, multi-directional latent space policy. Because multi-directional navigation is already learnt, the hypothesis is that it would be easier to learn navigation on top of locomotion. We perform an experiment to test this idea.

We trained the low-level policy with a reward to maximize velocity. We are able to reproduce and even surpass the results reported in (Haarnoja et al., 2018a) (see figure fig. 5).

An important assumption that is made in the hypothesis is that the behaviors that are learnt by the low-level policy are *diverse*. This is essential because the higher level policy accesses the environment via the low-level policy. If the low-level policy does not exhibit behaviors that are crucial to performing the higher level task, it might take a long time for the agent to learn. Additionally, (Haarnoja et al., 2018a) justify using soft actor-critic (SAC (Haarnoja et al., 2018c)) because it encourages learning of diverse behaviors by simultaneously maximising entropy of actions.

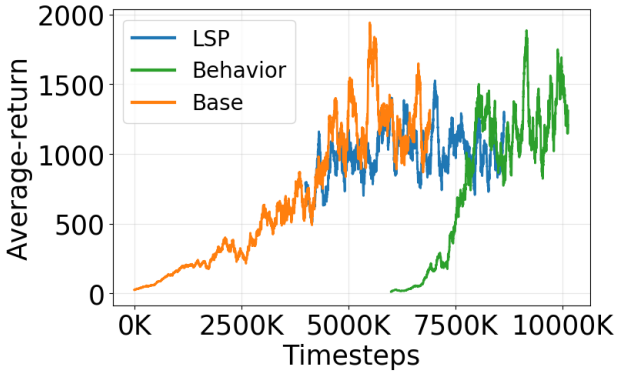



*Figure 8.* Training curves on Humanoid. The starting point of behavior policy curve and high level policy curve denote the number of low-level policy has been trained.

However, we found that the learned behavior of the agent is not diverse. We visualize the behavior of the trained agent and show it in figure fig. 4. We find that the agent has successfully learned to attain high velocites. However, it *only moves in a certain fixed direction*. On repeated rollouts of the policy with varying random seed, the agent consistently proceeds to move in the left or bottom-left direction. This shows that low-level policies are not learning diverse behaviors as we would have expected. Hence it is not clear how and where gains from the hierarchical latent space framework reported in Haarnoja et al. (2018a) come from.

### 4.2. Training with pre-trained behavior policy

In this experiment, we compare the performance a one level policy, a multi-level policy and our proposed one level policy trained with behavior policy on Humanoid environment 8. The low-level policy is a one level LSP trained from scratch, as shown in orange. The two level policy is trained by stacking another layer of LSP onto a trained policy after a certain number of training steps with the low-level policy frozen, as shown in blue. The one level LSP is training a pretrained one level LSP as behavior policy. The behav-

ior policy is used in the following way – during the first $1M$ steps, before the start of each episode we sample a Bernoulli random variable, if the result is 1, we execute the behavior policy at the current episode to collect trajectory, otherwise we execute the current policy at the current episode. The switch between behavior policy and current policy is necessary because without it, the $Q$ function network will only be exposed to actions with high rewards and therefore will output high value for all actions since it cannot distinguish between good actions and bad actions. In this experiment, we use the exact same number of layers and hidden sizes for all levels of policy and Q functions. We observe that with the help of behavior policy, our policy converges quickly and outperforms its behavior policy and the high level policy whose low-level policy is the same as the behavior policy. This shows that the benefit of using stacking multiple levels of LSP comes from a good behavior policy not from the transformation of environment.

## 5. Conclusion

In this work, we investigate the source of benefit of stacking multiple layers of latent space policies. We provide two interpretations of stacking multi-level LSP. One is that a low-level policy transforms the environment, such that the new MDP is easier to solve for a high level policy. The other is that low-level policy provides a good behavior policy to high level policy. In our experiments we try to disentangle the two factors, and perform attributions of performance gains. We show that better convergence and performance due to stacking higher level policies comes from good behavior policy and not from easier transition dynamics. In support of this, we present a theoretical analysis of why, under certain assumptions, it becomes easier to learn the optimal policy when a pre-trained policy is used as a behavior policy. Additionally, we perform a qualitative analysis of the types of behaviors learnt by low-level policies. We show that the low-level policy exhibits a surprising lack of diversity, which could be potentially detrimental when learning higher-level policies on top of it.

## References


Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Daniel, C., Neumann, G., and Peters, J. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pp. 273–281, 2012.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1352–1361. JMLR. org, 2017.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018a.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6244–6251. IEEE, 2018b.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018c.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science*, 313 (5786):504–507, 2006.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, Q. and Wang, D. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pp. 2378–2386, 2016.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Todorov, E. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, pp. 1369–1376, 2007.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. 2008.