HIERARCHICAL SKILL DISCOVERY VIA LATENT SPACE LEARNING

BENJAMIN TSOU

Abstract. An important area of reinforcement learning is learning skills in the absence of explicit environment rewards. In this work, we propose an unsupervised algorithm, *Hierarchical Latent Space Learning* (HLSL), that is able to learn a hierarchy of latent skills in an unsupervised manner. To train the skill policies, a hierarchical recurrent VAE model is used. Lower levels of the hierarchy focus on learning skill primitives. Higher levels of the hierarchy build upon these primitives to learn more temporally extended skills. An implementation of HLSL is available at `https://github.com/ben-tsou/hlsl`

1. Introduction

Deep reinforcement learning (RL) methods have proven capable of becoming highly proficient (and even producing super-human level performance) in various domains such as board games [26, 27], video game environments [18, 30], and robotic simulation environments [19, 24]. In each of these cases, a clear reward function is given and the algorithm is able to use the supervisory signal to learn and make progress. In environments such as the Atari game Montezuma's revenge in which rewards are extremely sparse (or even non-existent) and in which temporally extended tasks must be learned, standard RL methods have had little success. Recently, there has been interest in "unsupervised" RL, whose goal is to learn useful policies and behaviors when an explicit reward is unavailable.

One of the main approaches to unsupervised RL is unsupervised skill discovery [11, 10, 25, 16]. Here, instead of learning a policy $\pi(a_t|s_t)$ that maximizes a given reward function, a skill policy $\pi(a_t|z, s_t)$ is learned that represents a whole space of policies conditional on the latent code z in the skill space $\mathcal{Z}$. Practically speaking, these skills could then be used in a variety of downstream tasks, either by fine tuning the skills or using them as primitives in a hierarchical RL framework. From a more theoretical perspective, learning this space of skills can also be thought of as a form of representation learning for the environment, the way unsupervised ML techniques like the variational autoencoder (VAE) encode a dataset (e.g. images or sentences) in a latent space.

In the hierarchical RL (hRL) framework, several levels of policies are learned. For instance, with two levels, a higher level "manager" policy will pick lower level skills to run until a termination condition is reached, at which point a new skill is picked. Conceptually, the "manager" policy operates at a higher level of temporal abstraction and learns to pick skills that will help accomplish the task. The actual execution of the skills is then offloaded to the lower level skill policy. Assuming that coherent low-level policies have been learned, only a few steps of higher level policies need to be taken to reach sparse rewards, reducing the effective horizon of the task. This framework was largely introduced by Sutton [29] who used the term options instead of skills. Methods in hRL include the option-critic architecture [2] and HiPPO [17].

In this work, we propose an unsupervised algorithm named *Hierarchical Latent Space Learning* (HLSL) that is able to learn a hierarchy of latent skills. Thus, instead of only learning one space of latent skill primitives that can later be used in a supervised hRL framework, complex temporally extended skills are learned in an unsupervised manner. To the author's knowledge, this is the first method that attempts such a fully "unsupervised hRL" approach.

The action space for a skill $z \in \mathcal{Z}_L$ of level L is the latent skill space, $\mathcal{Z}_{L-1}$, of level $L-1$. Importantly, once an "action" or sub-skill in $\mathcal{Z}_{L-1}$ is picked, it is kept fixed for a number of steps (until a termination condition is reached). A common issue in hRL is skill collapse in which the lower level skills that are learned are trivial. Methods of addressing this include the notion of a "deliberation cost" to penalize switching skills too frequently [13] or optimizing the compressibility of the skills' encoding [14]. One of the simplest approaches is to use fixed length skills and this is the approach of our unsupervised method HLSL.

The collection of latent skill spaces $\mathcal{Z}_L$ (where L is the level of the hierarchy) is trained using a hierarchical recurrent VAE model that encodes trajectories as skills in latent space. The levels of the hierarchy are trained in a cyclical fashion so that they benefit from each other's training. Low level primitives that have been discovered can be used to assemble higher level skills. Conversely, higher level, temporally extended skills can reach more distant regions of the environment state space so that more robust lower level primitives can be learned.

For each level of the hierarchy, sample trajectories are drawn according to the currently learned skill spaces. To encourage exploration, these trajectories are also injected with some randomness. Then the VAE is trained for a few steps to encode the most salient features of the trajectories. With each iteration, alternating between sampling and training, more discernable skills are learned as the latent skill space takes shape. Examples of trajectories learned for the Cartpole environment are available in `sample_trajectory_videos` of the github repo.

2. Hierarchical Model

To accomplish the goal of learning coherent spaces of latent skills $\mathcal{Z}_L$, we train a hierarchical recurrent VAE model. Here, an overview of the model is given. The model consists of a VAE trained for each level L of the hierarchy. For experimental details, see Section 6.

2.1. **Primitive Level.**

At the lowest level, a VAE is trained that learns a space of "primitive" skills, $\mathcal{Z}_1$. Conceptually, these policies are simple mappings from state space to action space that can be built upon to create more complex skills. The VAE consists of an encoder $q^{\text{prim}}(z|x)$ and a decoder $\pi^{\text{prim}}(a_t|z, s_t)$. Here, the input x is a sequence of state action pairs $((s_1, a_1), ..., (s_T, a_T))$ that represent actions that a coherent policy $\pi(a|s)$ would take. Note that the states are not consecutive states in a trajectory, but rather a subset of representative states that might be encountered in a trajectory. Given this sequence of data, $q^{\text{prim}}(z|x)$ attempts to encode the information into a latent skill z such that the decoder policy $\pi^{\text{prim}}(a_t|z, s_t)$ can reconstruct these data points. In practice, $q^{\text{prim}}(z|x)$ is implemented with an RNN architecture and $\pi^{\text{prim}}(a_t|z, s_t)$ with a simple feed-forward network.

2.2. **Level 2.** The next level $L = 2$ of the hierarchy consists of an encoder $q_2(z|\tau)$ and decoder $\pi_2(a_t|z, s_t)$. Here, $\tau = (s_1, ..., s_T)$ is a trajectory of states. The encoder has a two level structure implemented with a bottom RNN and a top RNN. First, partition the trajectory into subtrajectories $\boldsymbol{\tau} = (\tau_1, \tau_2, ..., \tau_K)$. Each subtrajectory τ_i is passed through the bottom RNN g_2^{btm} to produce a final hidden state h_i. Then the sequence of hidden states $(h_1, ..., h_K)$ is fed as input into the top RNN g_2^{top} to produce a final hidden state h_{fin}. Finally h_{fin} is passed through two fully-connected layers to produce the VAE latent distribution parameters μ and σ. Following the "reparametrization trick", we then sample $\epsilon \sim \mathcal{N}(0, I)$ and form the latent vector $z = \mu + \sigma \odot \epsilon$.

The decoder policy $\pi_2(a_t|z, s_t)$ also has a two level structure. Given the skill $z \in \mathcal{Z}_2$, it selects a latent skill $z' = f_2(z, s_t)$ from $\mathcal{Z}_1$ every N_2 time steps when $t \equiv 0 \pmod{N_2}$ and runs the policy $\pi^{\text{prim}}(a_t|z', s_t)$ for the N_2 time steps in between.

2.3. **Level 3 and Higher.** Higher levels of the hierarchy build naturally on $L = 2$. For each level L, we have the encoder $q_L(z|\tau)$ and decoder policy $\pi_L(a_t|z, s_t)$. For the encoder, we define a new bottom encoder RNN g_L^{btm} and a new top encoder RNN g_L^{top} while for the decoder, we define a new mapping $f_L(z, s_t)$.

To construct the encoder $q_L(z|\tau)$, we process the trajectory sequence τ a total of $L-1$ times using the bottom encoder RNNs g_i^{btm} for $i = 2, ..., L$ before applying the top encoder RNN g_L^{top}.

The first time through, τ is processed by g_2^{btm} as described above to create the sequence of hidden states $(h_1^2, ..., h_{K_2}^2)$. This sequence is then partitioned into subsequences and processed by g_3^{btm} (on each subsequence) to form a sequence of hidden states $(h_1^3, ..., h_{K_3}^3)$. In general, the ith step of the process partitions the output of the previous sequence $(h_1^{i-1}, ..., h_{K_{i-1}}^{i-1})$ into subsequences, which are then passed through g_i^{btm} to form a sequence of hidden states $(h_1^i, ..., h_{K_i}^i)$. Finally, $(h_1^L, ..., h_{K_L}^L)$ is fed as input into g_L^{top} to produce the final hidden state h_{fin}^L, which is passed through fully-connected layers to produce the VAE parameters μ and σ.

The decoder hierarchical policy $\pi_L(a_t|z, s_t)$ can be defined simply via recursion. It selects a latent skill $z' = f_L(z, s_t)$ from $\mathcal{Z}_{L-1}$ every N_L time steps when $t \equiv 0 \pmod{N_L}$ and runs the policy $\pi_{L-1}(a_t|z', s_t)$ for the N_L time steps in between. Thus, the higher level policy networks deterministically output a latent code in the lower hierarchy latent spaces until finally the policy π^{prim} chooses a base action in the environment. For simplicity, the number of steps to take, N_L, before switching the latent sub-skill was kept constant rather than depending on the skill or state.

3. Learning Algorithm

The learning of the hierarchical VAE model operates by training levels one at a time (keeping the parameters of the other levels fixed) in an alternating fashion.

Initially, all the parameters are random, so the primitive level networks q^{prim} and π^{prim} are trained without using the other networks. Once the primitive level has been trained once, level $L = 2$ is trained using the crude primitives that have been learned thus far. Then training between the primitive level and 2nd level alternate until stable spaces of skills $\mathcal{Z}_1$ and $\mathcal{Z}_2$ have been learned. At this point, level $L = 3$ is trained using the previously learned skills. Training then alternates between the 3 levels trained so far until stable spaces of skills $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$ have been learned. This process can be continued to learn higher and higher levels of skills.

The training of a given level L of the hierarchy involves three steps. First, a skill selection procedure is used to decide which latent skills in $\mathcal{Z}_L$ to sample trajectories from. Then trajectories of various types are sampled based on these chosen skills. Finally, these trajectories are fed into the recurrent VAE for training. For each level of training, these three steps are repeated in an iterative loop.

The following gives an overview of the sampling and training process. Details of the procedure are given in Section 6.

3.1. **Skill Selection.** First, a set of N latent skills $z \in \mathcal{Z}_L$ are sampled according to the prior $p(z) \sim \mathcal{N}(0, I)$. For each z, a trajectory τ is sampled according to the policy $\pi_L^{\arg\max}(a_t|z, s_t)$ in which the argmax action of the policy $\pi_L(a_t|z, s_t)$ is always taken. Then each skill is given a weight proportional to $\exp(\mathrm{KL}(q(z|\tau)||p(z)))$. The motivation for this choice comes from exponentiating the evidence lower bound (ELBO) from the VAE (see equation 3.1). Recall that the ELBO gives a lower bound for the log likelihood of the data. Since the "data" in this case is essentially the output of our model, we can identify them and think of the ELBO as an approximation of the log likelihood. Then our choice of weight is essentially the reciprocal of the likelihood with the reconstruction loss term removed.

This choice of weighting makes it so that each "pattern" of trajectory is weighted equally. For instance, consider a VAE trained on a dataset consisting of 1000 handwritten digits of '0' and 500 handwritten digits of '1'. Assume the encoder is able to cleanly classify a handwritten digit into one of the two classes (a categorical prior over the set {0, 1} with probabilities {2/3, 1/3}) and the decoder is able to replicate the distribution of digits given the class. Then this weighting scheme will double the weight of class '1' so that both classes will be sampled equally.

Thus, as new "patterns" are discovered that are represented in the latent skill space with low frequency, they will be oversampled to encourage their incorporation in the latent skill space. In environments like Cartpole, the trajectories in which the pole falls over quickly are easy to learn, but are considerably less interesting. Thus, trajectory length is also incorporated into the weight to induce longer trajectories.

3.2. **Trajectory Sampling.** For each selected latent skill $z \in \mathcal{Z}_L$, both "argmax" and "repulsion" trajectories are sampled. Argmax trajectories are simply those sampled according to the policy $\pi_L^{\arg\max}(a_t|z, s_t)$. They represent what the encoder and policy networks have learned thus far. "Repulsion" trajectories are "argmax" trajectories in which random skills are spliced in at certain time steps. This helps to encourage exploration and prevent the training from degenerating. The initial state of the sampled trajectories is also varied so that policies learn coherent skills starting from a diverse set of states. This is done by first running the policy for a period of time and taking the resulting end state as the initial state of the sampled trajectory.

The sampled "argmax" and "repulsion" trajectories are also filtered by length to encourage the learning of longer trajectories.

3.3. **Training Loss.** Once all the trajectories have been sampled and filtered, the final curated set is sent to the hierarchical recurrent VAE model for training. For level L of the hierarchy, the loss to be optimized is the evidence lower bound (ELBO)

$$\mathbb{E}_{q_L(z|\tau)}\left[\sum_t \log \pi_L(a_t|z, s_t)\right] - \beta\ \mathrm{KL}(q_L(z|\tau)||p(z)) \tag{3.1}$$

where $\tau = (s_1, a_1, ..., s_T, a_T)$. The hyperparameter β is a standard modification used to adjust the weighting between the reconstruction loss and KL divergence terms of the VAE [5]. All parameters of the policy and encoder networks are kept fixed except for the level that is being trained.

4. Related Work

A similar recurrent VAE architecture was used in [5] to learn a latent space for sentence generation. More recently, this same architecture was applied to sheet music to create a "Music VAE" model [21] and to hand drawings to create a "SketchRNN" model [12]. The "Music VAE" model used a hierarchical decoder (but non-hierarchical encoder). To the author's knowledge, HLSL introduced in this work is the first instance of a hierarchical VAE structure being used in a reinforcement learning setting to learn a hierarchy of skills.

A number of previous works have studied unsupervised skill discovery. VIC [11] and DIAYN [10] attempt to learn discriminable skills while DADS [25] also encourages skills to be predictable according to the environment dynamics. In VALOR [1], it was shown that these methods can all be interpreted as a form of VAE training, except with the data space and latent space reversed. In other words, the skill space is encoded into trajectories, which is then decoded back into the skill space.

Methods such as SeCTAR [9], EDL [8], IBOL [16] learn diverse skills by training a VAE-like model in the more intuitive direction (in which the skill space is the latent space). The most recent algorithm IBOL [16] first learns a "linearizer" that abstracts away low level details of the environment dynamics by essentially learning a goal-conditioned low-level policy. Then a skill policy is learned on top that outputs goals for the low-level policy. This can be thought of as a two-level policy, but the skill policy that is learned outputs a goal every time step instead of keeping it fixed over a horizon. Thus, it's not hierarchical in the sense of hRL policies following the options framework. The method HIDIO [31] is a 2-level hRL method in which the upper level is trained on environment rewards and the lower level is self-supervised using intrinsic rewards.

In addition to skill discovery, there have been other approaches to incorporating intrinsic motivation [23, 3] into RL as a way to guide exploration. These methods tend to replace the extrinsic reward in the environment with various notions of intrinsic reward to help learn a single policy that will better explore the environment. Classes of exploration methods include count-based exploration [4, 7], curiosity-driven exploration [20, 6, 22], adversarial self-play [28], and information gain [15].

5. Experimental Notes

HLSL was implemented on a modified Cartpole environment in which some termination thresholds were increased to allow more room for skill discovery. Namely, `theta_threshold_radians`, the angle threshold, has been increased to $\pi/4$ and `x_threshold`, the x-position threshold, has been increased to 1000. The videos in the github repo show a sample of learned skills in $\mathcal{Z}_3$. Qualitatively, one can see that diverse and coherent skills are learned.

Sometimes, the model can get stuck in a local optimum in which relatively uninteresting skills are learned. Thus, a few runs may be required to produce desirable skills.

6. Algorithm Details

6.1. **Model architecture.** A 3-level hierarchical recurrent VAE structure was implemented for the Cartpole environment. The latent skill spaces $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$ of all hierarchical levels were chosen to be two-dimensional.

The encoders networks q^{prim}, g_2^{btm}, g_2^{top}, g_3^{btm}, g_3^{top} are all implemented as (single-layer) bidirectional LSTMs with hidden state size of 12. The LSTM for q^{prim} takes as input both state and action (concatenating the vectors) at each time step while the higher level encoder LSTMs only take the state s as input.

The decoder networks f_1, f_2, and f_3, for policies π_1, π_2, and π_3 are all 3-layer MLP feed-forward networks with 50 units per layer and ReLU activation function. They take in the skill latent vector z concatenated with the state s at each time step. There are environments (say music composition) where memory is important to a coherent policy, but this seems like a useful inductive bias in robotic environments where many interesting skills can be learned by memory-less policies. Also, strong decoders have the potential to suffer from posterior collapse and not store relevant information in the latent code.

For all networks, the x-position variable of the cartpole is removed from the state s before feeding in to encourage the learning of skills that are translation invariant. This technique was also used in DADS [25] and IBOL [16].

For q^{prim}, sequences of 20 state action pairs are fed in.

For the 2nd level, g_2^{btm} takes in subtrajectories of length 7 while g_3^{top} takes in subsequences of length 10.

For the 3rd level, g_3^{btm} takes in subtrajectories of length 7 while g_3^{top} takes in subsequences of length 10.

6.2. **Latent Space Weighting.** To come up with the latent skills $z \in \mathcal{Z}$ that are used for trajectory sampling, a grid of 400 latent codes $z_1, z_2, ..., z_{400}$ are first drawn according to equally spaced percentiles of the cdf of $\mathcal{N}(0, I)$. For each z_i, trajectories τ_i are sampled according to the policy $\arg\max_{a_t} \pi_L(a_t|z, s_t)$. For level $L = 2$ of the hierarchy, z_i is sampled with weight

$$w_i = \exp(\text{KL}(q_L(z_i|\tau_i)||p(z))) \cdot \max(T_i - 50, 0)^2$$

while for level $L = 3$ of the hierarchy, z_i is sampled with weight

$$w_i = \exp(\text{KL}(q_L(z_i|\tau_i)||p(z))) \cdot \max(T_i - 350, 0)^2$$

where $p(z) = \mathcal{N}(0, I)$ is the prior and T_i is the length of trajectory τ_i. Note that the maximum length for level $L = 2$ is $10 \times 7 = 70$ steps while for level $L = 3$, the maximum length is $10 \times 7 \times 7 = 490$ steps.

6.3. **Primitive Level Sampling.** The first time training the encoder $q^{prim}(z|x)$ and decoder $\pi^{\text{prim}}(a|s, z)$ for the primitive level of the hierarchy, the higher level networks have not been trained yet either. Thus, trajectories are sampled using only the decoder policy $\pi^{\text{prim}}(a|s, z)$. Let $z^* \in \mathcal{Z}_1$ be a latent skill code to be used for trajectory sampling. First, 20 independent "filler" codes $z'_1, z'_2, ..., z'_{20}$ are sampled from $\mathcal{N}(0, I)$. Form the sequence of latent codes $(z_1, ..., z_{40}) = (z^*, z'_1, z^*, z'_2, ..., z^*, z'_{20})$

that alternates z^* with the filler codes. Sample a trajectory by taking 5 steps according to the policy $\arg\max_{a_t} \pi^{\text{prim}}(a_t|z_i, s_t)$ from each latent skill, resetting the environment if a terminal state of the environment is reached. Then 20 equally spaced state action pairs (s_i, a_i), $i \in \{1, ..., 20\}$ are taken from the sections of the trajectory where the skill z^* is being used.

After higher levels of the hierarchy have been trained, the hierarchical decoder can be used to sample trajectories for primitive training. A mix of trajectories are sampled, some only running level $L = 2$ of the hierarchical policy $\pi_2^{\arg\max}(a|z, s)$ and some running the top level $L = 3$ of the hierarchical policy $\pi_3^{\arg\max}(a|z, s)$. A total of 600 steps are taken, where the environment is reset after each policy run. Then 20 state action pairs (s_i, a_i) are chosen, some randomly and some taken nearer the end of each run (close to the done condition) so that more precarious states are well represented.

Given these 20 state action pairs, four types of sequences are sampled: argmax, high repulsion, medium repulsion, and low repulsion. For argmax, no modifications are made. For the repulsion types, the policy probabilities $\pi^{\text{prim}}(a_t|z, s_t)$ of the argmax action are sorted from high to low. For high repulsion, the top third of actions are flipped (from 0 to 1 or from 1 to 0 since Cartpole has binary action space). For mid repulsion, the middle third of actions are flipped and for low repulsion, the bottom third of actions are flipped.

6.4. **Higher Level Trajectory Sampling.** For levels $L = 2$ and $L = 3$ of the hierarchy, the trajectory corresponding to latent skill $z^* \in \mathcal{Z}_2$ or $z^* \in \mathcal{Z}_3$ is sampled by running the hierarchical policy $\pi_L^{\arg\max}(a_t|z^*, s_t)$. Repulsion trajectories $\pi_L^{\text{rep}}(a_t|z^*, s_t)$ are also sampled (three times as many repulsion as argmax trajectories are sampled). The initial state of the trajectories are also varied by running the policy $\pi_2^{\arg\max}(a_t|z', s_t)$ or $\pi_3^{\arg\max}(a_t|z', s_t)$ first on a random filler code z' and taking the final state from this run as the initial state of the sampled trajectory.

For $L = 2$, the hierarchical policy $\pi_2^{\arg\max}(a_t|z^*, s_t)$ samples a latent code $z \in \mathcal{Z}_1$ every 7 steps and runs $\pi_1^{\arg\max}(a_t|z, s_t)$ for 7 steps until either a terminal state is hit or 70 steps are taken. $\pi_2^{\text{rep}}(a_t|z^*, s_t)$ is similar except that to encourage exploration, every other latent skill is taken to be a random $z' \in \mathcal{Z}_1$.

For $L = 3$, the hierarchical policy $\pi_3^{\arg\max}(a_t|z^*, s_t)$ samples a latent code $z \in \mathcal{Z}_2$ every $7 \times 7 = 49$ steps and runs $\pi_2^{\arg\max}(a_t|z, s_t)$ for 49 steps until either a terminal state is hit or 490 steps are taken. $\pi_3^{\text{rep}}(a_t|z^*, s_t)$ is similar except that to encourage exploration, every other latent skill is taken to be a random $z' \in \mathcal{Z}_2$.

After all the trajectories have been sampled, they undergo a length filtering process. Only the top half of trajectories by length are kept. The trajectories are further filtered to adjust the ratio of argmax and repulsion trajectories. The lower the median length of sampled trajectories, the fewer argmax trajectories are kept relative to repulsion trajectories to encourage exploration (details available in the file `LSL_utils.py`).

6.5. **Training.** For training the recurrent VAEs (at all hierarchical levels), an Adam optimizer with learning rate of 5e-3 and batch size of 8 were used. To encourage learning of policies that lead to longer trajectories, a negative "tail" weight is put on the reconstruction loss term for trajectories that end prematurely (don't have max length). For level 2 training, the last 7 actions receive negative weight in the reconstruction loss while for level 3 training, the last 25 actions receive negative weight.

Each training session is split into two parts. During the first part, both argmax and repulsion trajectories are sampled and KL cost annealing ([5]) of the β weight in front of the KL divergence term is used with a logistic learning schedule. During the second part, only argmax trajectories are sampled and no KL cost annealing is used.

6.6. **Cyclic Training of Hierarchies.** First, the primitive encoder $q^{prim}(z|\tau)$ and decoder $\pi^{prim}(a|s,z)$ are trained. Then (fixing these two networks), encoders q_2^{btm}, q_2^{top}, and decoder network $f_2(s,z)$ are trained.

Then level 1 and level 2 of the hierarchy are trained alternately 4 times (holding the other level fixed).

Then q_3^{btm}, q_3^{top} of level 3 of the hierarchy are trained along with the decoder network $f_3(s,z)$.

Finally, level 1, level 2, and level 3 of the hierarchy are trained cyclically a total of 4 times.

References

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms, 2018. https://arxiv.org/abs/1807.10299.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016.

[3] Andrew G. Barto. *Intrinsic Motivation and Reinforcement Learning*, pages 17–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[4] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation, 2016. https://arxiv.org/abs/1606.01868.

[5] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2016.

[6] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning, 2018. https://arxiv.org/abs/1808.04355.

[7] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018. https://arxiv.org/abs/1810.12894.

[8] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills, 2020. https://arxiv.org/abs/2002.03647.

[9] 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, 2018. https://arxiv.org/abs/1806.02813.

[10] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018.

[11] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control, 2016. https://arxiv.org/abs/1611.07507.

[12] David Ha and Douglas Eck. A neural representation of sketch drawings, 2017.

[13] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option : Learning options with a deliberation cost, 2017. https://arxiv.org/abs/1709.04571.

[14] Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic, 2019. https://arxiv.org/abs/1902.09996.

[15] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration, 2016. https://arxiv.org/abs/1605.09674.

[16] Jaekyeom Kim, Seohong Park, and Gunhee Kim. Unsupervised skill discovery with bottleneck option learning, 2021. https://arxiv.org/abs/2106.14305.

[17] Alexander C. Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning, 2019. https://arxiv.org/abs/1906.05862.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. https://arxiv.org/abs/1312.5602.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. 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.

[20] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction, 2017. https://arxiv.org/abs/1705.05363.

[21] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music, 2019.

[22] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability, 2018. https://arxiv.org/abs/1810.02274.

[23] Jürgen Schmidhuber. Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity & creativity. In Vincent Corruble, Masayuki Takeda, and Einoshin Suzuki, editors, *Discovery Science*, pages 26–38, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[24] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015. https://arxiv.org/abs/1502.05477.

[25] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills, 2020.

[26] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[27] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. https://arxiv.org/abs/1712.01815, 2017.

[28] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play, 2017. https://arxiv.org/abs/1703.05407.

[29] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[30] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[31] Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options, 2021. https://arxiv.org/abs/2101.06521.

Princeton, NJ 08540
Email address: `benjamintsou@gmail.com`