Skills as Gradients in Latent Space

Adam Klein
aklein4@stanford.edu

I. Abstract

This paper introduces the Skills as Gradients in latent Space (SAGS) unsupervised skill discovery algorithm. Unsupervised skill discovery is a reinforcement learning method that seeks to train agents to act in environments without reward functions. This is done by maximizing the mutual information $I(z;x)$ between the choice of skill z, and some metric about the trajectory, x. This is done by creating a model to calculate either $p(z|x)$ or $p(x|z)$.

Previous methods for unsupervised skill discovery have used metrics that include the states s visited in the trajectory (in DIAYN [1]) and the transitions from one state to another, (s', s) (in DADS [2]). SAGS builds upon these works by introducing a new metric that can be correlated with skills: latent space transitions.

Consider two states, s_1 and s_2. Given that we have an encoder that encodes those states into latent representations l_1 and l_2 respectively, the latent transition can be described as the difference between those two representations: $\Delta l = l_2 - l_1$.

SAGS uses this latent transition as the metric for its mutual information maximization: $I(z;\Delta l)$. This is beneficial compared to using the states themselves because latent spaces can have an element of explainability. Furthermore, latent transitions can cover multiple steps, rather than a single step as other methods are bound to. Finally, hierarchical reinforcement learning (HRL) algorithms have already seen success in using latent space transitions to inform agents [3]. In order to maximize the mutual information, SAGS trains a model to compute $p(z|\Delta l)$ and a policy with a reward equal to $\log p(z|\Delta l)$.

SAGS also introduces a new concept to unsupervised skill discovery: combining skills. Here, we look at how humans can take simple skills and combine them to form more complicated behaviors. For this, we introduce the notion of skill conditions: instead of choosing one skill to act on at a time, each skill i can either be in an active condition z_i or inactive condition $\neg z_i$. Then, since we are concerned with the value $p(z|\Delta l)$, we define a combination of skills in terms of their joint probability $p(z_1 + z_2|\Delta l) = p(z_1 \cap z_2|\Delta l)$.

To further specify the joint probability representation of skill combinations, we assume that skill conditions are sampled independently. This allows us to represent the total log probability of a skill as a linear combination of the log probabilities of each of the constituent skills: $\log p(z|\Delta l) = \log p(z_1|\Delta l) + \log p(z_2|\Delta l)$.

Then, we introduce an attention term α_i over the constituent skills. This keeps the magnitude of skill combinations consistent across different numbers of constituent skills. It also provides a continuous parameter space for skill combinations. The attention also simplifies implementation because instead of sampling discrete subsets of skills to train on, we can mask out irrelevant skills using low attention. This gives us the final joint skill representation: $\log p(z|\Delta l) = \sum \alpha_i \log p(z_i|\Delta l)$.

Given our joint skill probability representation, we also derive a path integral interpretation of latent transitions through a skill space. This allows us to break trajectory-wise $\Delta l_{start,end}$ based rewards into step-wise $\Delta l_{t,t+1}$ based rewards. This makes our algorithm much more robust to sample collection methods and more sample efficient.

We then detail an algorithm to maximize our $I(z|\Delta l)$ target using a soft Q-learning [4] policy and a trained state encoding model. This algorithm is similar to those seen in DIAYN [1] and DADS [2], with the policy and encoder playing a collaborative game to optimize our target term. The algorithm includes a number of tricks to make it work better, such as only back-propagating into the encoder for s_{t+1}, not s_t.

Then, we describe a zero-shot motion planning algorithm for SAGS agents. This algorithm leverages the fact that we can easily compute Δl for two states. The motion planning works by applying Bayes' theorem to our learned $p(z|\Delta l)$ to get $p(\Delta l|z)$. We can then find the z' that makes Δl most likely. This algorithm also allows for in-the-loop control by refreshing Δl with every subsequent s_t.

In our experiments section, we discuss the performance of the SAGS algorithm on an environment with simple yet non-trivial dynamics. We train our latent space to be a function of only x and y coordinates which allows us to easily visualize the latent space.

In our results, we see that diverse skills are learned for both $n = 2$ skills and $n = 8$. These skills are shown to be described by the gradients of the latent fields, giving them a method of explanation. Furthermore, the learned skills are shown to combine in semantically meaningful ways, as was the goal of our joint skill representation.

Finally, we demonstrate the abilities of our motion planning algorithm using those skills. The agent is able to successfully navigate not only from the start state to target states, but along a path defined by way-points. This shows that skills learned by SAGS are useful and generalizable, and could be a useful interface between a high-level manager and a low-level worker.

In summary, our results point to SAGS being a powerful method for unsupervised skill discovery, learning skills that are generalizable, explainable, and useful. Future work could study how SAGS skills work in an HRL setting, while also testing SAGS in more complicated environments.

II. Background

Reinforcement learning (RL) algorithms have seen great success learning to act in environments with predefined rewards. However, some environments have sparse rewards that can be difficult for agents to discover. Furthermore, reward functions can be difficult and expensive to calculate [1]. Because of this, there is an emerging field of reinforcement learning devoted to exploring environments in an unsupervised fashion, in order to discover skills.

Skills are policy conditions that lead to a predictable behavior. Learning skills can act as a pretraining task for a reinforcement agent. Skills can also be leveraged by hierarchical reinforcement learning (HRL) algorithms. In HRL, a high-level manager to instructs a low-level worker in how to act in the environment. This can be done by choosing skills for the worker. [3]

Most skill discovery algorithms rely on maximizing the the informativeness between the skill and what the agent does. This essentially works out to being a game of charades: the policy tries to act out the skill, while some other system is tries to figure out what skill it is. This can be expressed mathematically as maximizing the mutual information between the skill and some metric x about the trajectory it generated: $I(s; x)$ [1].

One of the most successful algorithms for unsupervised skill discovery, and the one most relevant to this paper, is Diversity Is All You Need (DIAYN) [1]. In DIAYN, the metric is the states that the agent visits. This means that DIAYN is trying to optimize $I(z; s)$, by learning to predict the skill given the state, $p(z|s)$. However, in order to make skills more robust and meaningful, DIAYN found that we should also attempt to maximize the entropy of the policy, $H(a|s, z)$. This ensures that different skills must explore sufficiently different states in order to remain distinguishable.

Another skill discovery algorithm is Dynamics-Aware Unsupervised Discovery of Skills (DADS) [2]. DADS maximizes the mutual information between the skill and the next state given the current state, $I(s', z, s)$. To achieve this, DADS learns a model of the environment conditioned on the skills, $p(s'|z, s)$. This allows DADS to also use model-based reinforcement learning (MRL) techniques to perform zero-shot navigation of the environment.

III. Methods

A. SAGS Overview

The first component of this paper is an unsupervised skill discovery algorithm that builds upon the information-based schemes discussed in DIAYN and DADS. Coined Skills As Gradients in latent Space (SAGS), this algorithm seeks to improve the generalizability, explainability, and usefulness of learned skills.

The concept for SAGS is inspired by the manager-worker interface of the FuN HRL algorithm [3]. In FuN, the high-level manager specifies tasks as *directions in a learned latent state space*. The goal of the lower-level worker then is to move through the latent space in the specified direction. SAGS takes this latent space learning and applies it to unsupervised skill discovery.

To define it what means to act in latent space, consider two states from our state space, $s_1, s_2 \in \mathcal{S}$, and function $L : \mathcal{S} \to \mathcal{L}$ that encodes states into the latent space $\mathcal{L}$. We define the *latent transition* Δl of s_1 and s_2 as the difference between the two states' latent representations:

$$\begin{aligned} &l_1 \in \mathcal{L} = L(s_1),\ l_2 \in \mathcal{L} = L(s_2) \\ &\Delta l = l_2 - l_1 \end{aligned}$$

With a well-defined latent representation, this transition could have an explainable semantic meaning relating increasing or decreasing properties of the state. For example, consider a latent representation of a robot's state in which the first dimension can be interpreted as how much the robot is 'walking'. Then, if $\Delta l^{(1)}$ is positive, Δl may be interpreted as 'starting to walk'.

They way that SAGS actually works is that it takes the framework of DIAYN and replaces state-based skill definitions with latent transition-based skill definition. In addition to linearity properties exploited later, the hope is that the latent transitions may be more meaningful to a trajectory than its visited states. Furthermore, latent transitions need not be single steps in the environment, and could encode an entire trajectory by taking $\Delta l = l_{end} - l_0$. The concept of using latent transitions to direct an agent has also already been demonstrated by FuN [3].

To discover skills relating to latent transitions, SAGS has three optimization goals based on those of DIAYN [1]:

1) Maximize the mutual information $I(\mathcal{L}; \mathcal{Z})$ between our latent space $\mathcal{L}$ and our skill space $\mathcal{Z}$. This means that given a latent transition, we can predict which skill was most likely to have generated it.
2) Maximize the entropy $\mathcal{H}[\mathcal{A}|\mathcal{S}]$ of the policy. As discussed in DIAYN, this ensures that skills must explore sufficiently different locations in the state space (latent space in the case of SAGS) in order to remain distinguishable [1].
3) Minimize the mutual information $I(\mathcal{A}; \mathcal{Z}|\mathcal{L})$ between our action space $\mathcal{A}$ and our skill space $\mathcal{Z}$. This makes sure that it is actally the latent transitions that are used to predict skills, not actions.

Together, these goals give us our optimization function $\mathcal{F}(\theta)$:

$$\begin{aligned} \mathcal{F}(\theta) &:= I(\mathcal{L}; \mathcal{Z}) + \mathcal{H}[A|s] - I(\mathcal{A}; \mathcal{Z}) && (1) \\ &= \mathcal{H}(\mathcal{Z}) - \mathcal{H}(\mathcal{Z}|\mathcal{L}) + \mathcal{H}(\mathcal{A}|\mathcal{Z}, \mathcal{S}) && (2) \end{aligned}$$

DIAYN also shows how we can use Jenson's Inequality to get a variational lower bound on this objective [1]:

$$\begin{aligned} \mathcal{F}(\theta) &= \mathcal{H}(\mathcal{A}|\mathcal{Z}, \mathcal{S}) - \mathcal{H}(\mathcal{Z}|\mathcal{L}) + \mathcal{H}(\mathcal{Z}) && (3) \\ &\geq \mathcal{H}(\mathcal{A}|\mathcal{Z}, \mathcal{S}) + \mathbb{E}_{z\sim p(z), \Delta l \sim \pi(z)}[\log q_\phi(z|\Delta l) - \log p(z)] && (4) \end{aligned}$$

The first element of our lower bound is the policy entropy, which is maximized using by using a soft Q-learning policy [4]. The $\log p(z)$ element can be minimized by using a uniform distribution when sampling z. Therefore, most of this methods section is devoted to learning and optimizing the function $\log q_\phi(z|\Delta l)$.

B. Combining Skills

The second component of this paper addresses a core concept to human skill abstraction that is often overlooked by skill discovery methods: combining skills. As a motivating example, consider the task of turning in a car. We have two skills: steer the wheel, and engage the turning signal. Instead of learning an entirely new skill, a human would be able to 'add' the two known skills to get a new one suited to the task:

$$\text{Turn} = \text{Steer} + \text{Signal}$$

However, in previous works, skills are usually trained independently and abstractly; There are no guarantees for how skills may behave when combined, or even how to combine them. Therefore, our goal is to define and learn skills that are not only useful on their own, but can also be combined in a semantically logical way to create *new skills*.

To this end, we take a close look at what skills actually represent. In DIAYN, they represent a probability distribution $p(z = i|s)$ over states, such that for a given state s there is some probability that we using the skill z_i [1]. SAGS is similar, except with latent transitions replacing states.

Given this description, what should it look like to combine skills? Consider a set of states $z_i \in \mathcal{Z}$. Rather than assuming that one skill is being used at a time, we say that each skill has either an active, z_i, or an inactive, $\neg z_i$, condition. Relating to SAGS, we are concerned with the likelihood of a skill's condition given latent a transition: $p(z_i|\Delta l)$. Therefore, a reasonable way to define a combination of skills is the joint probability of their conditions:

$$p(z_1 + z_2|\Delta l) = p(z_1 \cap z_2|\Delta l),$$
$$p(z_1 + \neg z_2|\Delta l) = p(z_1 \cap \neg z_2|\Delta l)...$$

This means that making the combined skill probable requires making both component skills probable. To see that this makes sense, we return to the car turning example. Here, we represent the probability that we are turning as the probability that both the steering *and* signalling skills are active.

$$p(\text{Turn}|\Delta l) = p(\text{Steer} \cap \text{Signal}|\Delta l)$$

C. Joint Skill Representation

Given our joint probability definition for skill combinations, how can we learn skills that actually work that way? First, we assume the conditions of skills to be conditionally independent over the latent transition. This means that for 2 skills, z_1 and z_2, the probability that z_1 is active depends entirely upon the current state, and our belief does not change when z_2 becomes active. We ensure this independence by sampling from $p(z)$ independently.

$$p(z_1|\Delta l, z_2) = p(z_1|\Delta l)$$

The joint probability can then be represented using the product of each skill. This is useful because it then makes our joint log probability linear with respect to each skill.

$$p(z|\Delta l) = \prod p(z_i|\Delta l)$$
$$\log p(z|\Delta l) = \sum \log p(z_i|\Delta l)$$

However, this representation becomes problematic during learning, especially if there are many skills. This is because if there are many skills that are being considered, then the joint probability will be very low, even if each individual probability is relatively high.

To address this problem, we introduce an attention coefficient α_i for each skill z_i:

$$p(z|\Delta l) = \prod p(z_i|\Delta l)^{\alpha_i} \quad (5)$$
$$\log p(z|\Delta l) = \sum \alpha_i \log p(z_i|\Delta l) \quad (6)$$

If $\sum \alpha_i = 1$, then our total likelihood will retain its magnitude as more skills are added. This has an added benefit of being a continuous parameter space, giving more precise control and acting as a powerful interface for a down-the-line hierarchical manager. This also simplifies training by allowing us to keep every skill together (rather than sampling subsets), with low attention masking out the skills we don't care about.

It is not yet understood how the sampling method for the attentions affects the the the prior probability of z, so for now it is ignored in the prior calculation:

$$p(z) = \prod p(z_i)^{\alpha_i} \quad (7)$$
$$\log p(z) = \sum \alpha_i \log p(z_i) \quad (8)$$

D. Learning Joint Skills with SAGS

Given our descriptions of SAGS and joint skills, we need a method for optimizing our variational lower bound (4) using them. Since the first term is optimized using soft Q-learning, and $\log p(z)$ is optimized using a uniform distribution, we are left with the optimization target:

$$\mathcal{G}(\theta) = \mathbb{E}_{\Delta l \sim \pi(z)}[\log q_\phi(z|\Delta l)] \quad (9)$$

The most obvious way to optimize $\mathcal{G}(\theta)$ is to simply train a discriminator to represent $\log q_\phi(z|\Delta l)$ as a function of Δl, propagating into our encoder L. However, we want Δl to be able to cover trajectories rather than single steps, and with this method we would be left with with a single terminal reward to train our policy on. This would make our algorithm more difficult to train and less robust to sampling methods. Therefore, we need a way to break $\log q_\phi(z|\Delta l)$ into step-sized pieces that can be applied as individual rewards.

It turns out that we can get our step-wise reward function by treating $\log q_\phi(z|\Delta l)$ as a line integral through our latent space. To do this, consider the energy function $\mathcal{E}_{A,C} : \mathcal{L} \to \mathbb{R}$

parameterized by our attentions $A \in \mathbb{R}^n$ and our skill conditions $C \in \mathbb{R}^n$ (n is the number of skills):

$$C_i = \begin{cases} 1 & z_i \\ -1 & \neg z_i \end{cases}$$

Let this energy be a linear function of our position in the latent space:

$$\mathcal{E}_{A,C}(l) = (A \odot C)^\top l$$

Then, let $\epsilon_{A,C} : \mathcal{L} \to \mathbb{R}^n$ be equal to the gradient of $\mathcal{E}_{A,C}$ (this assumes that $\mathcal{L}$ has dimension n, which can be achieved using a linear projection if not already true):

$$\epsilon_{A,C}(l) = \nabla \mathcal{E}_{A,C}(l) = (A \odot C)$$

Now, let $\log q_\phi(z|\Delta l) = \mathcal{E}_{A,C}(l_2) - \mathcal{E}_{A,C}(l_1)$. This satisfies our joint probability definition with $\log q_\phi(z_i|\Delta l) = C_i \Delta l_i$:

$$\begin{aligned} \log q_\phi(z|\Delta l) &= \mathcal{E}_{A,C}(l_2) - \mathcal{E}_{A,C}(l_1) \\ &= (A \odot C)^\top l_2 - (A \odot C)^\top l_1 \\ &= (A \odot C)^\top \Delta l \\ &= \sum_{i=1}^{n} \alpha_i C_i \Delta l_i \\ &= \sum_{i=1}^{n} \alpha_i \log q_\phi(z_i|\Delta l) \end{aligned}$$

Now, consider the latent path $l_0, l_1, ...l_t$ for the trajectory $s_0, s_1, ...s_t$ with $\Delta l = l_t - l_0$. Since $\epsilon_{A,C}(l) = \nabla \mathcal{E}_{A,C}(l)$, the fundamental theorem of calculus for line integrals gives us that $\log q_\phi(z|\Delta l)$ is equal to the line integral of the path through $\epsilon_{A,C}(l)$ (note that while this is a discrete sum instead of an integral, the identity holds since $\epsilon_{A,C}(l_i)$ is constant).

$$\begin{aligned} \sum_{i=0}^{t-1} \epsilon_{A,C}(l_i) \cdot \Delta l_i &= \sum_{i=0}^{t-1} \nabla \mathcal{E}_{A,C}(l_i) \cdot \Delta l_i \\ &= \mathcal{E}_{A,C}(l_t) - \mathcal{E}_{A,C}(l_0) \\ &= \log q_\phi(z|\Delta l) \end{aligned}$$

Now, we can represent our optimization target (9) in the typical reinforcement learning manner, with step-wise rewards:

$$\mathcal{G}(\theta) = \mathbb{E}_{s \sim \pi(z)} \left[\sum_{t=0}^{T} \epsilon_{A,C}(l_i) \cdot \Delta l_i \right] \tag{10}$$

$$= \mathbb{E}_{s \sim \pi(z)} \left[\sum_{t=0}^{T} (A \odot C)^\top \left(L(s_{t+1}) - L(s_t) \right) \right] \tag{11}$$

$$r(s_t, s_{t+1}) = (A \odot C)^\top \left(L(s_{t+1}) - L(s_t) \right) \tag{12}$$

Unfortunately, $\log q_\phi(z|\Delta l)$ is bound to the interval $(-\infty, 0]$, while $(A \odot C)^\top \Delta l$ is open to all of $\mathbb{R}$. To remedy this, we approximate using the following (which holds for small $C_i \Delta l_i$):

$$(A \odot C)^\top \Delta l \approx A^\top \log \sigma(C \odot \Delta l) + \log(2) \tag{13}$$

There are other variations that can be used for this approximation, such as putting A^t inside the sigmoid, but this version seems to work as well as any. Since $A^\top \log \sigma(C \odot \Delta l)$ is bound to the interval $(-\infty, 0]$, we ensure that $\log q_\phi(z|\Delta l)$ does not go above zero. Furthermore, the $\log(2)$ bias term can be ignored for the purpose of optimizing a fixed length trajectory. While I could rewrite the preceding proof to incorporate the intervals and bias, I will instead point to my results section to show that this substitution seems to work in practice.

$$\mathcal{G}(\theta) = \mathbb{E}_{s \sim \pi(z)} \left[\sum_{t=0}^{T} A^\top \log \sigma(C \odot (L(s_t) - L(s_{t+1})) + \log(2) \right] \tag{14}$$

$$r(s_t, s_{t+1}) = A^\top \log \sigma(C \odot (L(s_t) - L(s_{t+1})) + \log(2) \tag{15}$$

Note above that the bias is still included in the reward function, as it helps center the rewards around zero.

E. The SAGS Algorithm

```
Algorithm 1: SAGS
Initialize $L$, $\pi$, empty $\mathcal{B}$;
while not converged do
    Sample a new skill $z \sim p(z)$ every episode;
    Collect $\mathcal{D}$ on-policy samples;
    $\mathcal{B} \leftarrow \mathcal{D}$;
    train $\pi$ on $r(s_t, s_{t+1}) \in \mathcal{B}$;
    train $L$ on $J_L(s_t, s_{t+1)} \in \mathcal{D}$;
end
```

Now that we have a step-wise optimization target (14) and reward function (15), we can present the full SAGS algorithm. The algorithm contains 2 parameterized models:

1) The state encoder: $L : \mathcal{S} \to \mathcal{L}$
2) The policy $\pi(a|s, z)$ represented by a soft Q-learning model.

The policy is trained to optimize the reward function (15), while the state encoder is trained using gradient descent on a loss function that directly follows from (14). This forms a collaborative game between the two models. It has also been found that applying L2-regularization to Δl is essential to avoiding divergent behavior. This can be interpreted as keeping states that are fewer steps away from each other closer in the latent space, and is important for maintaining the linearity assumption in (13).

$$\begin{aligned} J_L = - \sum_{A,C,s_t,s_{t+1} \in \mathcal{D}} \Big[& A^\top \log \sigma(C \odot (L(s_{t+1}) - L(s_1)) \\ & - \lambda_L ||(L(s_{t+1}) - L(s_t)||_2 \Big] \end{aligned}$$

Furthermore, it has been found to be crucial that gradient is *only* passed into L through $L(s_{t+1})$, *not* through $L(s_t)$. Another possible problem is that upon initialization, Δl is either too large or too small. In that case, a static scale should be applied such that the elements of Δl span the range $(-1, 1)$. Finally, it has been found that the encoder should have a slow learning rate, approximately $1/100$ that of the policy, or else the reward function will change too fast for the policy to adjust.

As previously stated, the policy is trained using soft Q-learning [4]. Skills are fed into the network by concatenating $A \odot C$ onto the state. Both a periodically refreshed target network and experience an replay buffer were used in our experiments to improve the policy's performance. For entropy, a value of $\alpha = 0.25$ was found to work well. An empirical tip for training the policy is that it helps if rewards are given a static scale to bring their average absolute value to around 1.

For sampling z from $p(z)$, entropy is maximized by sampling C_i from $\{-1, 1\}$ with equal probability. Sampling the attention A is a little trickier, with seemingly equal results from $\alpha_i \sim \text{Exp}(\lambda = 1)$ (focusing on fewer skills) and $\alpha_i \sim \text{Uniform}(0, 1)$ (higher entropy). In both cases, A is L1 normalized after sampling.

For collecting samples from the environment, the method used in this paper was to initialize every episode from the same state and use the same z for the entire episode. DIAYN samples initial states [1], but it is unclear how this would effect SAGS performance.

The final thing to point out is that while the policy uses an experience replay buffer, the encoder only trains on the most recent samples. Furthermore, it seems to help if the policy's rewards are calculated before the encoder has been updated on the most recent samples, to avoid bias.

F. SAGS Motion Planning

The SAGS algorithm provides one more feature that makes its learned skills exceptionally useful: zero-shot motion planning capabilities. Suppose that we are in some s_0 and want get to some s_{target}. We can directly calculate the latent transition for these two states as $\Delta l = L(s_{target}) - L(s_0)$. We can then easily calculate the probability of a skill condition given that latent transition using our learned estimator:

$$\log q_\phi(z|\Delta l) = A^\top \log \sigma(C \odot \Delta l) + \log(2)$$

Now, in order to have the highest likelihood of moving to s_{target} from s_0, we just need to find the skill condition z that maximizes $\log p(\Delta l|z)$. For this, refer to the log-space Bayes' rule:

$$\log p(\Delta l|z) = \log q_\phi(z|\Delta l) + \log p(\Delta l) - \log p(z) \qquad (16)$$

Since $\log p(z)$ is constant over all z, we can discard it for the purpose of optimization. We can also disregard $\log p(\Delta l)$ since we already know our Δl. Then, substituting our estimator into the remaining term gives:

$$A', C' = \arg\max_{A,C} A^\top \log \sigma(C \odot \Delta l)$$

Finding C' is easy: $C'_i = \text{sign}(\Delta l_l)$. Furthermore, given the L1-normed restriction on A, we have A' as a one-hot vector at the largest element of $\log \sigma(C' \odot \Delta l)$ [5]. However, in practice, this doesn't seem to capture the direction of the latent transition very well. Instead, empirical results point to the following as the best values for A and C:

$$C'_i = \text{sign}(\Delta l_l),\ A' = \frac{\log \sigma(C' \odot \Delta l)}{|| \log \sigma(C' \odot \Delta l)||_1} \qquad (17)$$

Furthermore, the skill condition can be updated at each step to achieve in-the-loop control to correct for drift. For this, we simply recalculate A' and C' at each t using $\Delta l = L(s_{target}) - L(s_t)$.

Algorithm 2: SAGS Motion Planning

```
Initialize $s_0, s_{target}$;
while not arrived do
    $\Delta l \leftarrow L(s_{target}) - L(s_t)$;
    $\forall 1 \le i \le n : C_i \leftarrow \text{sign}(\Delta l_i)$;
    $A \leftarrow \frac{\log \sigma(C' \odot \Delta l)}{|| \log \sigma(C' \odot \Delta l)||_1}$;
    sample $a \leftarrow \pi(a|s_t, A, C)$;
    $s_t \leftarrow$ step environment with $a$
end
```

This type of control has the interesting property that the agent's goal is to minimize the distance between s_t and s_{target} in the *latent space*, rather than the real state space. It is difficult to predict the full implications of this property, but for complicated state spaces, latent distance could be a very useful metric for tracking agents' progress.

IV. Experiments

A. Experimental Setup

The SAGS algorithm was tested in a specially designed environment, which we refer to the 'amoeba' environment. This environment consists of a 2d plane inhabited by a character. The character has a position (x, y) and a direction θ. Position is bound to the range $(-1, 1)$, and direction is on the range $(-\pi, \pi)$. The state is represented by:

$$s = \big[x, y, \cos(\theta), \sin(\theta)\big]$$

There are 9 discrete actions that can be taken, each corresponding to a combination of moving forward/backward from the set $\{-0.02, 0, 0.02\}$ and turning left/right from the set $\{-\pi/20, 0, -\pi/20\}$. At each step, turning is applied first, then movement. Position is clipped into the defined range when exceeded.

This environment was used because its simple yet non-trivial dynamics allow test to be ran quicker than in 3d simulation environments. Furthermore, the 2d plane makes evaluation easier. Similar to in the DADS paper, we train the encoder on only the (x, y) position [2], so that we can interpret the latent space more easily.

B. Results: Learned Skills

The system was trained to convergence until the average $\log q_\phi(z|\Delta l)$ value converged.. First, with only two skills.

In figure 1 we see the latent space projections of the two skills. Notice that the gradients of the two skills are orthogonal to each other. This means that we can describe any direction in the (x, y) plane as a combination of skills.

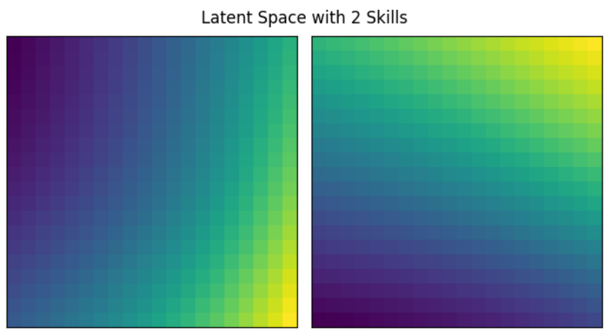

Fig. 1: Latent space encodings projected onto the (x,y) plane with $n = 2$.

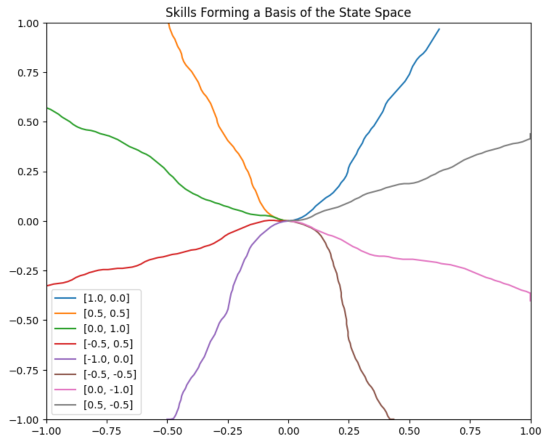

Fig. 2: Agent paths with different skill inputs when $n = 2$. The legend represents $(A \odot C)$ of the skills.

Figure 2 shows trajectories taken by the agent with different skill conditions and attentions. We see that the skills seem to form an orthogonal basis for the agent's directions, and line up with the gradients seen in figure 1. This means that skills learned by SAGS are actually gradients in the latent field, as we described them to be in the Learning Joint Skills with SAGS section above. This gives us a powerful method for visualizing and explaining skills.

In figure 3 we see example trajectories taken by the agent when interpolating attention between the two skills. Rollouts were done greedily to increase clarity. The two skills seem to combine in a semantically logical way, with the 'Northeast' skill combining with the 'Northwest' skill to form a 'North' skill.

However, it may be possible that having 2 skills in a simple 2d state space is a special case that makes SAGS work particularly well. Therefore, we also tested a version with 8 skills, to force more interesting skills to be learned.

Figure 8 in the appendix shows the latent space projections for the 8 learned skills. We see that the gradients of the skills point in mostly different directions. However, some of the skills converged to being nearly identical. This could be a byproduct of having too many skills for the simple state space.

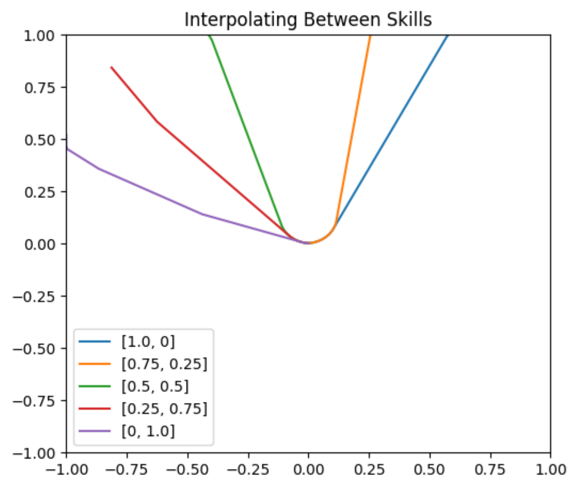

Fig. 3: Agent paths interpolating between the two skills when $n = 2$ with greedy execution. The legend represents $(A \odot C)$ of the skills.

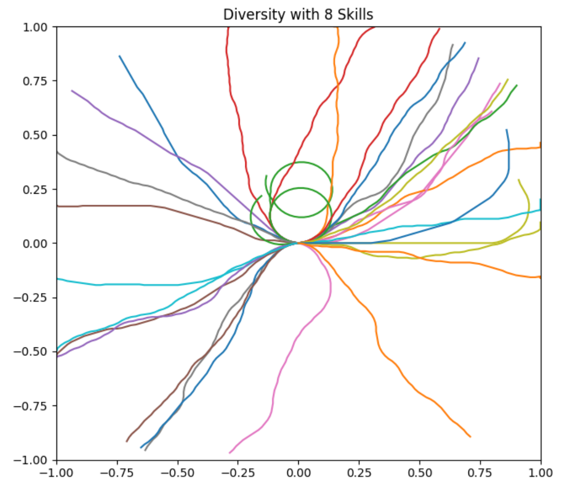

Fig. 4: Agent paths sampled from 64 random skill conditions and attentions, with $n = 8$.

In figure 4 we see 64 trajectories for randomly sampled skill conditions and attentions with $n = 8$. There is a large amount of diversity in the agent's directions, visiting almost every part of the state space.

Figure 5 shows trajectories interpolated between the first 2 skills with $n = 8$, again sampled greedily for clarity. Again we see that the skills combine in a semantically meaningful way, with direction being interpolated as the skills are interpolated.

These results show that our algorithm works as intended. The agent learned a diverse set of skills that can be combined in semantically meaningful ways. Furthermore, the skills can be explained using their latent space projections. The generalizability and usefulness of the skills are explored further in the next section.

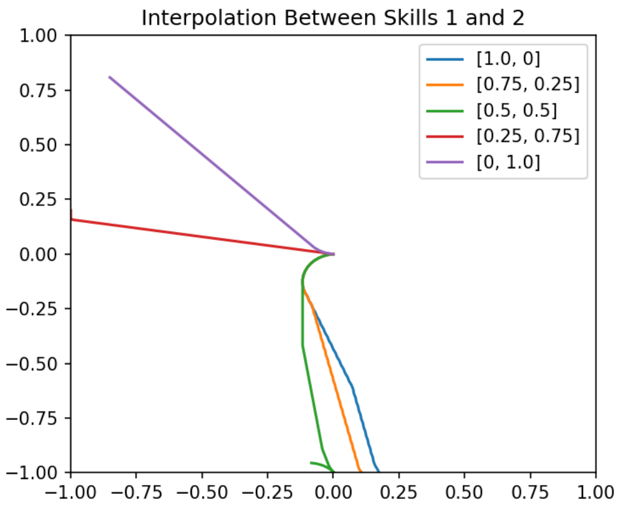

Fig. 5: Agent paths interpolating between the first two skills when $n = 8$ with greedy execution. The legend represents $(A \odot C)$ of the skills.

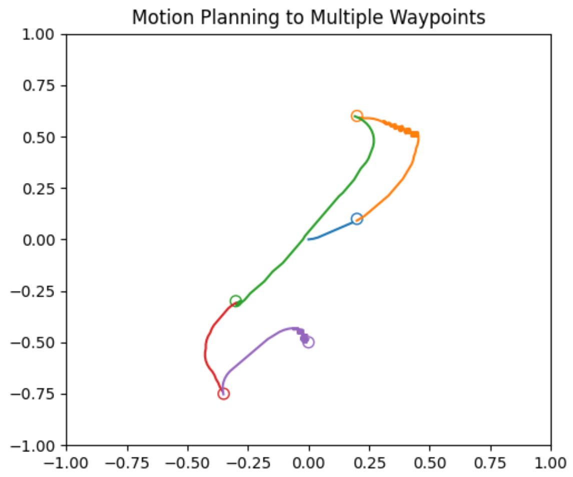

Fig. 7: Agent trajectory following a path defined by way-points (circles). Rollout used entropy of $\alpha = 0.01$.

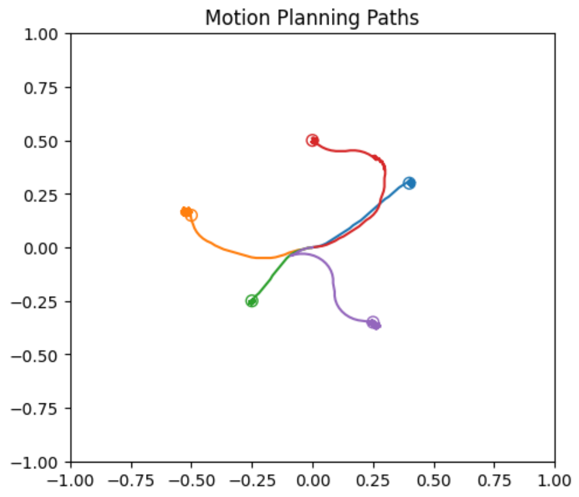

Fig. 6: Agent paths using in-the-loop path planning to move from the starting position to each of the target locations (circles). Rollouts used entropy of $\alpha = 0.01$.

C. Results: Motion Planning

Using the learned skills above, we also tested the abilities of the SAGS Motion Planning Algorithm. The following tests were done using the $n = 8$ skills seen above, to avoid the special two-skill/two-dimension case described previously.

First, in figure 6, we see the results of the agent using in-the-loop planning to move to different target states from the start position. We see that the agent successfully moves to the target state before stopping there. It should be noted that the success of the agent varied greatly with the entropy of the policy. Rollouts with the same $\alpha = 0.25$ entropy from training tended to do poorly, while greedy rollouts performed better but could get stuck in places. The best results were achieved with $\alpha = 0.01$.

To show the generalizability of the learned skills, figure 7 shows the agent navigating a path defined by way-points. The agent started in the start state and navigated to the first way-point. Once it was within a small distance, the target switched to the next way-point, and so on. The success of the agent on this test shows that skills are not bound to the starting state, and generalize to other positions.

The motion planning results demonstrate the the skills learned by SAGS are useful, since they allow us to plan trajectories with no further training. Furthermore, the skills were shown to generalize and work in different positions.

V. Future Work

The next step in the development of SAGS would be to test it in more complicated environments, such as those seen in DIAYN and DADS. While SAGS was shown to work very well in a simple environment, it remains to see how well it scales.

Another extension to SAGS would be putting it into an HRL algorithm to see how well a higher-level agent could direct a worker that was trained using SAGS skills. The FuN algorithm already provides a framework for HRL using latent transitions [3] and could serve as a starting point.

Finally, the algorithm itself has open questions. What if we calculated $\log q_\phi(z|\Delta l)$ using a different equation or another model? What if we trained using random starting positions? How can we extend skills to low-frequency observations? Can we create a hierarchy of skills that operate on lower-level skills?

VI. Conclusion

In this paper, we described the SAGS algorithm for unsupervised skill discovery using latent space transitions. We showed that skills learned using SAGS that are generalizable, explainable, and useful. Furthermore, we showed that those skills can be combined in semantically meaningful ways, opening the door for complex behaviors that require only a few base skills. Finally, we demonstrated a zero-shot motion planning algorithm that allows SAGS agents to navigate to target states with no additional training.

VII. References

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

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

[3] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017.

[4] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017.

[5] David Meyer. Notes on maximization of inner products over norm balls, 2017.

VIII. Appendix

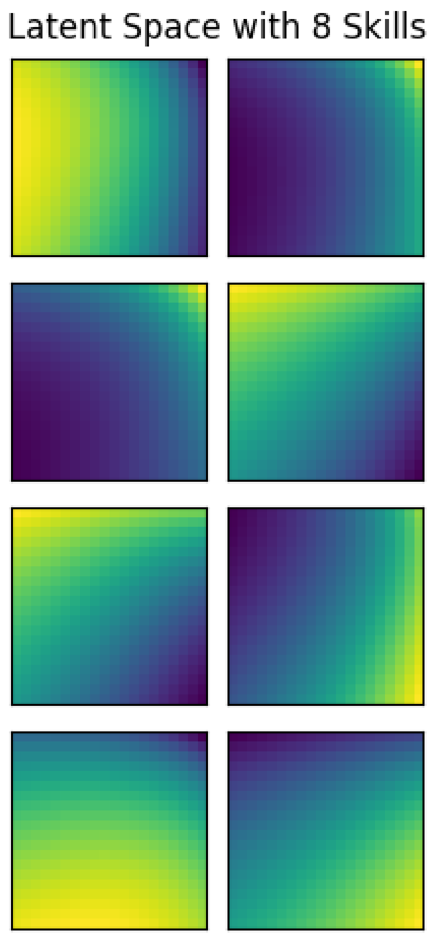

Fig. 8: Latent space encodings projected onto the (x,y) plane with $n = 8$.