General Characterization of Agents by States they Visit

Anonymous Author(s)
Affiliation
Address
email

Abstract

Behavioural characterizations (BCs) of decision-making agents, or their policies, are used to study outcomes of training algorithms and as part of the algorithms themselves to encourage unique policies, match expert policy or restrict changes to policy per update. However, previously presented solutions are not applicable in general, either due to lack of expressive power, computational constraint or constraints on the policy or environment. Furthermore, many BCs rely on the actions of policies. We discuss and demonstrate how these BCs can be misleading, especially in stochastic environments, and propose a novel solution based on what states policies visit. We run experiments to evaluate the quality of the proposed BC against baselines and evaluate their use in studying training algorithms, novelty search and trust-region policy optimization.

1 Introduction

While creating or training autonomous agents, whether it is via manual coding, *reinforcement learning* (RL) [49] or *evolution strategy* (ES) [18] algorithms, one wishes to compare the solutions to find out which work and which do not. A common approach is comparing the performance of agents' policies in the given task [20] or by studying how policy behaves [39]. In addition to these explicit comparisons, RL and ES algorithms include implicit comparisons to encourage finding novel solutions [29, 8, 12, 37], to match the behaviour of an expert policy [24, 35] or to limit changes on policy's behaviour to avoid catastrophic failure [45, 46].

To facilitate such comparisons, *behavioural characterizations* (BCs) [29] aim to capture policy's behaviour in a fixed representation accompanied by a distance metric. A general BC could be used to generalize previously proposed methods [8, 36]. However, BCs in the previous work are often domain-specific or depend on policy's actions which are not descriptive of behaviour in general, as we will discuss in Section 3.

Recent work attempted to generalize the notion of a *behavioural embedding* [36], considering policies as distributions over trajectories, and providing a principled mechanism to compare policies in the behavioural space. However, this approach relies on "behavioural embedding mappings", which map trajectories to structures that are believed to describe the trajectory (e.g., reward-to-go, terminal state). It remains unclear how the choice of this mapping affects the results.

In this work, we propose and evaluate a more general BC that can be applied to different domains. Our work provides the following contributions: (1) We summarize and compare BCs used in the previous work. (2) We discuss and demonstrate the shortcomings of action-based BCs which have been actively used in the previous work. (3) Propose a novel BC method based on which states the agent visits, borrowing techniques from the field of speaker recognition [28]. (4) We use the proposed

Submitted to Workshop on Deep Reinforcement Learning (NeurIPS 2021). Do not distribute.

Table 1: Comparison of different BCs. Required compute describes CPU and memory requirements for comparing a large set of policies, excluding the sampling of the environment. Environment agnostic methods do not pose requirements for the environment, and action agnostic methods can compare policies across action spaces.

Description and reference	Required compute	Describes	Expressiviness	Agnostic	
				Env.	Action
Returns [20]	Low	Policy	Low	✓	✓
Policy parameters [13]	Low	Policy	Low	✓	✗
Comparing actions [36, 19]	Low	Policy	Low	✓	✗
Termination state [8]	Low	Policy	Low	✗	✓
Transition matrix [32]	V.High	Policy	High	✗	✗
Trajectories [8]	High	Trajectory	High	✗	✓
States + Discriminator [12]	N/A	Skill [12]	High	✓	✓
States + Gaussian [3]	Low	Policy	Low	✓	✓
Trajectory encoder [52, 43]	High	Trajectory	High	✓	✗
Aggregate state-actions [17, 50]	High	Policy	High	✓	✗
States + Discriminator (Adapted)	Low	Policy	High	✓	✓
Trajectory encoder (Adapted)	High	Policy	High	✓	✓
States + GMM (Proposed)	Low	Policy	High	✓	✓

BC to study and visualize different training algorithms and also explore their use in novelty search [29] and trust-region policy optimization [45].

2 Preliminaries and definitions

We model environments as stochastic *Markov decision processes* (MDPs) [49], where agent acts on states $s \in \mathcal{S}$. The environment begins in an initial state $s_0 \sim p(s_0)$. The policy $\pi \in \Pi$ provides an action $a \in \mathcal{A}$ as a stochastic function $a \sim \pi(s)$. After executing an action, the environment evolves to the next state according to a stochastic function $s' \sim p(s'|s, a)$ (transition dynamics), and agent is provided with a reward $r \sim p(r|s, a, s')$. This process repeats until environment lands into a terminal state $s_T \in \mathcal{S}$. A single episode begins from an initial states and ends to a terminal state. A trajectory is a tuple of states and actions experienced during one episode. An environment is defined by tuple $(\mathcal{S}, \mathcal{A}, p(s_0), p(s'|s, a), p(r|s, a, s'), \mathcal{T})$.

Behaviour characterization is a function $b : \Pi \to \mathcal{B}$, where $\mathcal{B}$ is space of all behaviours, accompanied with a *pseudo-distance* function $d : \mathcal{B} \times \mathcal{B} \to \mathbb{R}^+$. This need not be a proper distance metric, as long it is non-negative and symmetric. Ideally, the more different behaviour of two policies is, the larger this distance should be. Note that b may be a stochastic function if, for example, it relies on randomly sampled data to map policy to a behaviour.

3 Choosing behavioural characterization

Table 1 compares various BCs used in the previous research, which have been either used explicitly for visualization (returns, terminal state) or implicitly in the training algorithm (e.g., novelty search [47], diversity [12]). Table 1 also includes methods which model trajectories or skills (e.g., fixed policy conditioned on a latent vector), rather than policies.

While we could use policy's parameters to compare agents, this does not generalize over different types of algorithms nor can we tell if a change in parameter vector is truly meaningful to the agent's behaviour [8, 19]. Episodic returns are the *de facto* approach for measuring quality of policies, but policies with distinct behaviour can achieve same returns (see Figure 1) [1]. Using a termination state, such as the final coordinates of the robot, has been successful in novelty search but is limited to environments where such heuristics can be used. Element-wise comparisons of trajectories require either fixed-length episodes or heuristics to combine varying length episodes. Finally, one can estimate transition dynamics by sampling the environments [32], but this has a high computational

cost that limits the number of policies compared. With these options discarded, we are left with BCs which focus on actions policy takes or states they visit.

3.1 Actions and stochastic environments

Comparing policies by what actions they chose is intuitively a sound solution, as one may argue the policy's actions are what define its behaviour. Action-based BCs are used in gradient-based RL algorithms to restrict changes to policy's behaviour [45, 46] or to encourage diversity [37]. This is done by comparing actions of policies they would take in a fixed set of states (an "off-policy embedding" [36]).

However, using actions alone for BC would ignore the transition dynamics of the environment. As discussed by Pacchiano et al. [36], a small change to actions could lead the agent to wildly different states or a large change in actions could have no effect.[1] To illustrate this, Figure 1 shows three scenarios where the behaviour of two policies are represented with the average episodic return, distribution of the taken actions and distribution of the visited states.

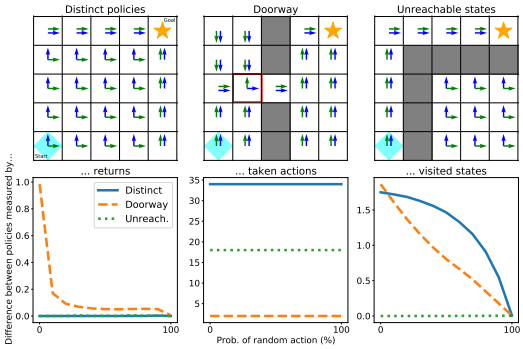


Figure 1: Difference between the two policies, blue and green, as measured by different BCs. State BC is obtained with ten thousand trajectories per policy, and distance is measured as the sum of the absolute errors over states. Action BC difference is the sum of the absolute difference between action distributions. Details are available in Appendix A.

Action-based BC is agnostic to environment stochasticity. As stochasticity of the environment increases, the policies tend towards random behaviour, and action-based BC describes them as two very different policies while they both behave the same (random agents). Another weakness is the insensitivity to "doorway" scenarios, where a single action can lead to different states. Action-based BC shows the two policies are very similar, but the green policy never reaches the other side of the world until we increase stochasticity. In real environments, doorway scenarios may manifest as literal doorways, critical points in grasping an object or pressing a button in a video game. Finally, even if some of the states used to compare action-differences were unreachable, they affect the results. In the right-most figure of Figure 1, both policies behave exactly the same in the area where they can traverse, yet the action-based distance is high.

In summary, while action-based BCs are useful for RL training, they can not be relied upon as a general BC: without accounting for the environment dynamics, one can not say if the difference in actions is meaningful.

3.2 Describing policies by states they visit

A common alternative [12, 3] to action-based BCs is comparing *what states policies visit*. This requires sampling states for each policy, but in turn, it captures the environment dynamics. Referring to Figure 1, where action-based BC provided misleading descriptions, state-based BC captures correctly the increasing stochasticity and the effect of the doorway and unreachable states. Compared to returns, it also captures the difference between two optimal but distinct policies.

A simple state-based BC is to fit a multivariate Gaussian on the states visited by the policy [3]. Both fitting and comparing policies is fast (e.g., KL-divergence, which can be evaluated in a closed-form), but limits the description to a unimodal distribution. The true distribution of states can be complex even for a simple environment (see scatter plots in Figure 2). Instead of a single Gaussian, a neural network discriminator can be trained to measure the probability of a state coming from a given policy, as done by Eysenbach et al. [12] and Ni et al. [35], Alternatively, one can train an auto-encoder to encode varying-length trajectories into fixed length [52]. Alas, the BCs of these works are tightly integrated with the training loop, and can not be used with policies trained by other algorithm. Instead,

[1] If an agent takes an action and it has no effect on the environment, did the agent take an action from an outside observer's point of view?

we propose using a mixture of Gaussian models to model complex distributions of states, which is detailed in the next section.

4 Policy supervectors

We consider a setup with N policies $\pi_i,\ i \in \{1 \dots N\}$ which we want to compare. For each policy, we play M episodes and store all encountered states $\boldsymbol{s}$ to a per-policy dataset $\boldsymbol{B}_i = (\boldsymbol{s}_{\{i,1\}}, \boldsymbol{s}_{\{i,2\}}, \dots)$. Note that the stored states may differ from states the policy acts upon; stored states can be a separate piece of information believed to describe the behaviour, such as x-y coordinates of a maze robot [39]. In the case of high-dimensional environments, one can train an encoder to turn images into more compact latent codes [3] or to preprocess them into low-dimensional representations [9].

Gaussian mixture models (GMMs) [33] can be used to model multi-modal data, and with enough components they can model almost any continuous distribution to an arbitrary accuracy [4]. However, measuring KL-divergence requires expensive sampling. While approximations exist, they come with different assumptions and drawbacks [22].

Instead, we draw inspiration from the field of speaker verification where GMMs have been used to model speaker characteristics [28]. Instead of training one GMM per speaker, Reynolds et al. [44] presented *universal background models* (GMM-UBMs, later UBMs) to model a distribution of speaker-independent features by fitting a large GMM on a large pool of speaker data. Speaker-specific model can then be adapted with *Maximum a Posteriori* (MAP) adaptation of GMM components [15] using individual speaker's data. The acquired parameters can then be concatenated into a long vector, called a *supervector* [7], describing the speaker with a fixed-length vector.

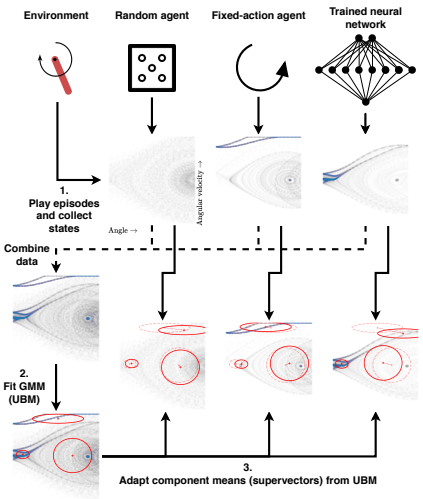


Figure 2: Computing policy supervectors for three policies in the Pendulum environment, using real data and policies. Each faint dot represents a single state of the Pendulum environment, and red circles represent Gaussian components of the GMMs.

We apply this method to policies instead of speakers. We first fit a GMM on state datasets from all policies and then MAP-adapt a supervector for each policy, which we call a *policy supervector*. Under the Bayesian view, the UBM represents an informative prior while the adapted GMM is a point estimate of the posterior, where we use the same data as part of the prior and adapting the posterior. We treat UBM as a common feature extractor, and by including data from all policies we ensure the prior covers all states visited by all of the policies. This process is illustrated in Figure 2.

Formally, policies' datasets are pooled together into an UBM-training set $\boldsymbol{B}_{\text{ubm}} = (\boldsymbol{B}_1, \dots, \boldsymbol{B}_N)$. This set is then used to fit an UBM with K components and parameters $\boldsymbol{\mu}_{\text{ubm}} \in \mathbb{R}^{K \times d}$, $\boldsymbol{\Sigma}_{\text{ubm}} \in \mathbb{R}^{K \times d \times d}$ and $\boldsymbol{w}_{\text{ubm}} \in \mathbb{R}^K, w_k \geq 0, \sum_k w_k = 1$ with EM-algorithm until convergence. We can then perform MAP-adaptation on a per-policy dataset $\boldsymbol{B}$ to obtain adapted mean $\hat{\boldsymbol{\mu}}_k$ of kth component with [44]

$$p(k|\boldsymbol{s}_t) = \frac{\boldsymbol{w}_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^{K} \boldsymbol{w}_l \mathcal{N}(\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} \tag{1}$$

$$n_k = \sum_{t=1} p(k|\boldsymbol{s}_t) \tag{2}$$

$$E_k(\boldsymbol{B}) = \frac{\sum_{t=1} p(k|\boldsymbol{s}_t)\boldsymbol{s}_t}{n_k} \tag{3}$$

$$\alpha_k = \frac{n_k}{n_k + r} \tag{4}$$

$$\hat{\boldsymbol{\mu}}_k = \alpha_k E_k(\boldsymbol{B}) + (1 - \alpha_k)\boldsymbol{\mu}_k, \tag{5}$$

where r, known as a *relevance factor*, is a control parameter that impacts how much new data can affect the adapted mean.

To measure the distance between two policy supervectors, Campbell et al. [7] show that an upper-bound for KL-divergence of means of two adapted GMMs $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ is

$$d_{\mathrm{KL}}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) \leq \frac{1}{2} \sum_{k=1}^{K} w_{\mathrm{ubm}}^k (\boldsymbol{\mu}_i^k - \boldsymbol{\mu}_j^k) \boldsymbol{\Sigma}_{\mathrm{ubm}}^{-1} (\boldsymbol{\mu}_i^k - \boldsymbol{\mu}_j^k) \ . \tag{6}$$

Unlike KL-divergence, this upper-bound is symmetric [7]. In this work, we will only adapt the means of the UBM to allow the use of the above metric, but this method can be extended by adapting covariances and weights (see [44] equations (11)-(13)). We use this upper bound as a distance metric for policies.

Limitations. Policy supervectors require each policy to sample data from the environment which can be expensive in slow environments. In highly stochastic environments we need more samples to accurately describe policies, which will make the training phase of these methods more demanding. Policy supervectors also require the environment to be the same for all policies, which can be difficult for real-world robotics, for example.

5 Adapted baselines from related work

Given the number of work towards characterising policies (Table 1), we adapt two of the previous work to form two baseline solutions. These methods share the same limitations as supervectors discussed above.

5.1 Discriminator as a state-density estimator

Motivated by the earlier use of discriminator networks as a density estimator [12, 35], we train one network per policy to distinguish between states this policy encounters versus all other policies. Specifically, we train a discriminator neural network D_π by ascending the loss

$$\mathcal{L}_i = \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{B}_i} \left[\log D_i(\boldsymbol{s}) \right] + \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{B}_{j \neq i}} \left[\log(1 - D_i(\boldsymbol{s})) \right] \ , \tag{7}$$

where i is the index of the policy we are about to compare to others. Optimal discriminator satisfies properties [16, 35]

$$D^*(\boldsymbol{s}) = \frac{p_i(\boldsymbol{s})}{p_i(\boldsymbol{s}) + p_{j \neq i}(\boldsymbol{s})} \tag{8}$$

$$\Rightarrow \frac{p_i(\boldsymbol{s})}{p_{j \neq i}(\boldsymbol{s})} = \frac{D^*(\boldsymbol{s})}{1 - D^*(\boldsymbol{s})} = r_i(\boldsymbol{s}) \ , \tag{9}$$

where $p_i(\boldsymbol{s})$ is shorthand for $p_{\boldsymbol{B}_i}(\boldsymbol{s})$. This value $r_i(\boldsymbol{s})$ represents how likely it is the state was observed by π_i than by the rest of the policies. With this in mind, we define a pseudo-distance between two policies,

$$f_i(\boldsymbol{s}) = e^{-\log r_i(\boldsymbol{s})} \tag{10}$$

$$d(i, j) = \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{B}_i} \left[f_j(\boldsymbol{s}) \right] + \mathbb{E}_{s \sim \boldsymbol{B}_j} \left[f_i(\boldsymbol{s}) \right] , \tag{11}$$

summing distance both ways for symmetry. Essentially, for each policy, we measure how likely it is that their data is also contained in others' datasets and vice versa. In practice, the expectations are computed as averages over the datasets. We opt for training one discriminator per policy (policy's states versus all others) rather than one per comparison (policy X's states versus policy Y's) to avoid the quadratic explosion of discriminator training operations required. Training details can be found in Appendix B.4.

5.2 Trajectory encoder

Wang et al. [52] proposed to train a *variational autoencoder* (VAE) [27] to reconstruct trajectories. The encoder turns a sequence of states (trajectory) into an embedding $\boldsymbol{z}$, and the decoder reconstructs original state-action pairs from this embedding. We adapt this method for policies by modelling

the distribution of trajectory embeddings. We train the system on data from all policies, then use the encoder to sample one embedding per trajectory, and then fit a multivariate Gaussian on these embeddings. The distance between two policies is then symmetric KL-divergence of these two Gaussians, where we sum the KL-divergences both ways. Further details are available in Appendix B.5.

Compared to the two other proposed methods, this method can capture temporal dependencies between states, while the other two methods discard this information. However, this could lead to posterior collapse where the decoder predicts future states purely based on previous states, ignoring the embedding. If the environment and/or policy are complex, we need a complex model to encode the whole policy into a fixed embedding. Finally, out of the three methods, training the encoder is the most computationally expensive.

6 Related work

The *de facto* method for comparing policies is by evaluating their performance in a task [25], or if the environment permits it, we can compare the different coordinates policy visits [39, 8]. Hernandez et al. [21] make use of dimensional reduction to plot fixed-length trajectories of different self-play algorithms to study the evolution of policies, and Matusch et al. [32] compare discretized transition matrices to study which metrics correlate with human behaviour. Others propose decompressing compact representations of policies back into functional policies, such as from genomes [13] or random generator seeds [48].

Trust-region policy optimization [45] limits per-update change to policy's behaviour to ensure improvement in returns, but similar methods can also be used to encourage novel behaviour [29, 8] or diverse behaviour [12, 37]. An RL agent can be encouraged to visit unseen states [2], where visit counts can be estimated by prediction error [6]. Framing imitation learning as a task of matching state or state-action distributions has also been successful [24, 35, 30]. Compact policy embeddings have also been used to create generalized value functions [43, 50, 19] or to improve imitation learning [52].

7 Experiments and results

For an empirical assessment of the proposed BC (policy supervectors), we first evaluate it against the baseline BCs and then explore their use in various tasks. We aim to answer the following research questions (RQs). **RQ1** Does the BC separate truly different policies (expressivity)? **RQ2** How sensitive are the BCs to the random sampling of the environment? If BC does not describe the policy, there is no reason to use it. If it produces wildly different results even with a lot of data, its results may not be trusted. Note that we have not included action-based BCs because of the reasons discussed in Section 3.1 (their measurements can not be trusted). However, we will compare to them in trust-region experiments later (Section 7.5).

We use five classic control environments from the OpenAI Gym library [5] which include low dimensional environments (Cartpole, Pendulum and Acrobot) as well as two higher-dimensional environments (Lunarlander and Bipedal-walker). We opt for these simple environments to allow scaling the experiments. We train three *proximal policy optimization* (PPO) [46] agents per environment and store 100 versions of the policy during training. For each policy, we then collect a set of trajectories, which we use to compute the state-based BCs and finally measure distances between these 100 policies. We repeat this data collection and distance measurement three times. Distance matrices are min-max normalized to $[0, 1]$ to allow comparison between different repetitions. Technical details, source code and further results described can be found in the Appendix B.

7.1 Baselines

In addition to adapted baselines (Section 5), we include two baseline solutions.

Discretization. Similar to Matush et al. [32], we discretize each state dimension to ten equally spaced bins, count occurrences of each bin and divide by the number of states visited. The distance between two policies is then $d(p_i, p_j) = \frac{1}{2} \sum_s |p_i(s) - p_j(s)|$. We opt to use this sum over KL-divergence due to the prevalence of zeros in the distributions. While simple, this method's computational requirements explode with the increasing number of dimensions.

Single Gaussian. We fit a multivariate Gaussian on the sampled states and define distance between two policies as $d_{\mathrm{KL}}(p_i||p_j) + d_{\mathrm{KL}}(p_j||p_i)$ for symmetry. This has been used previously in tracking what states policy has visited [3].

7.2 Evaluation metrics

Correlation with return-distance. To measure how well BC separates different policies, we measure the Pearson correlation between the absolute difference between average returns and distances measured by the BC. While returns are limited in their expressiveness, it still indicates the policies are doing something different if the returns differ. This especially applies to the environments we use, as the reward signal is tied to the states agent visits.

Distance error. Given a "ground truth" distance between policies under the same BC, we measure the average relative error between this ground truth and predicted distances. We select one of the repetitions with the highest amount of data (100 trajectories) to represent this ground truth. This metric aims to measure how sensitive the method is to the amount of data we collect.

Distance variance. As behavioural distances between a fixed set of policies should stay the same relative to each other, we study the variance in the results by computing *coefficient of variation* (CV) [11] σ/μ over the repetitions, where σ is the standard deviation of the distance over repetitions and μ is the sample average of distances. A lower value indicates more similar results over repetitions.

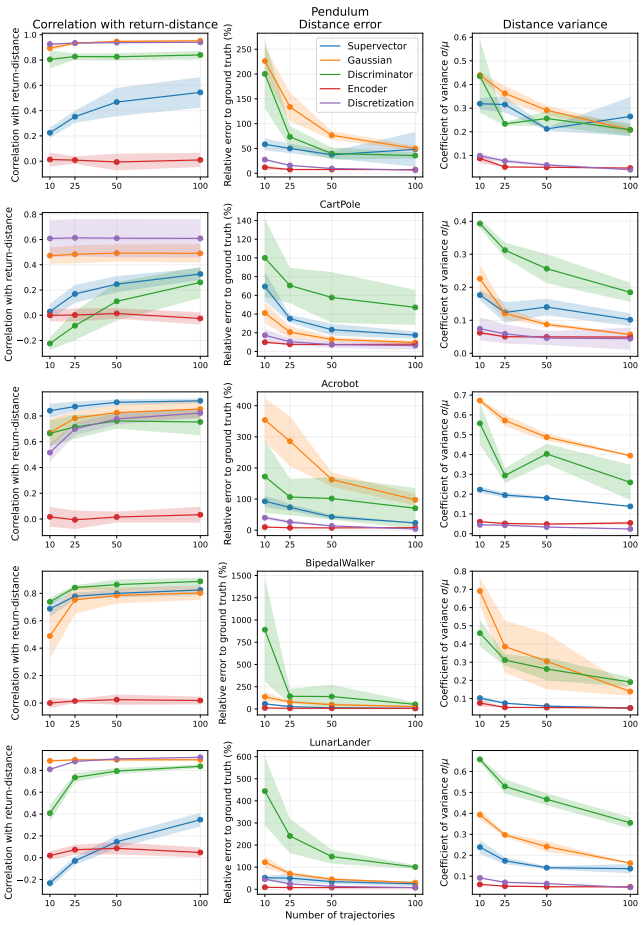


Figure 3: Evaluation results of different BCs. Averaged over three repetitions. The shaded region is plus/minus one standard deviation.

7.3 Evaluation results

Figure 3 shows the results for each environment separately. The single Gaussian and discriminator methods correlate with return differences, but require many trajectories per policy to stabilize results. Policy supervectors with 64 Gaussian components offer stabler results even at a lower number of trajectories, which we believe is due to the use of data from all policies in the UBM training. By sweeping over the different number of components, we find a strong connection between environments and the optimal number of components: in some environments one to four components provides the stablest results (Appendix B).

The discretization method appears to capture the behaviour as well as remain stable, but this method is computationally limited: experiments in the BipedalWalker environment ran out of memory (64GB of system memory) when trying to build the transition matrix over multiple dimensions. Trajectory encoders were similarly limited, which we found to be slow to train due to long sequences. We believe the low performance of encoder method is due to the difficulty of predicting future states. In summary,

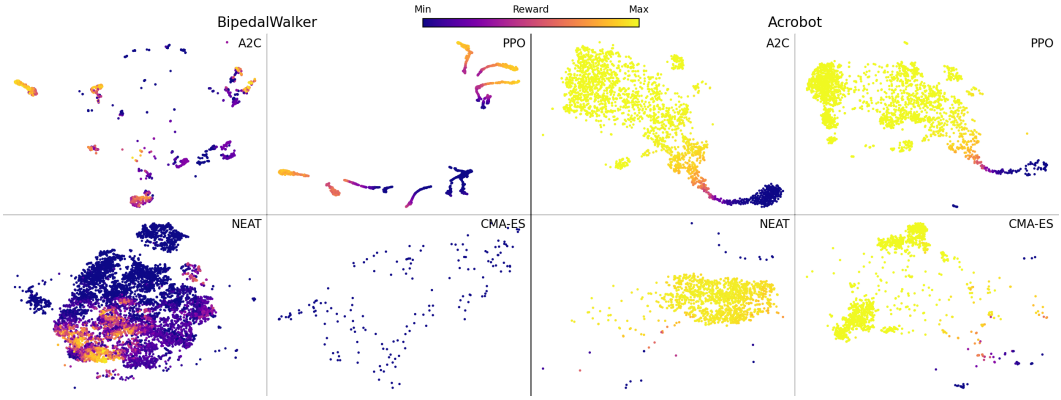


Figure 4: t-SNE plot of the evolution of policies under different training algorithms, where each point represents a single policy. Plots under the same environment share the same plot scales. Rewards are scaled according to minimal and maximal attainable reward per environment.

we find policy supervectors a scalable and functioning state-based BC, with discretization a viable option in low-dimensional environments but not generally applicable due to memory constraints.

7.4 Studying evolution of policies under training algorithms

Evolution-based methods such as *neuroevolution of augmenting topologies* (NEAT) [47] and *covariance matrix adaptation ES* (CMA-ES) explore by modifying promising solutions and testing which location of the parameter space works better [18], while gradient-based RL methods like *advantage actor critic* (A2C) [34] and PPO update a single policy towards higher episodic reward. To better understand how these training algorithms evolve policies, we train agents with them and store checkpoints during training. We then extract policy supervectors of these checkpoints (with 64 components) and plot the resulting points with t-SNE [31] dimensionality reduction using the adapted distance (6). The hypothesis is that evolution-based methods cover a wider area of policies (random mutation of the parameters leads to different behaviours), while RL algorithms remain in a small region per run. We are not comparing which of these algorithms is better, rather we aim to understand if they explore different behaviours and how they evolve the policies.

Figure 4 shows the results with five A2C/PPO runs and one NEAT/CMA-ES run. In total, roughly 30,000 policies are compared against each other, with a varying number of policies from different algorithms depending on the settings (see Appendix C for details and remaining plots). NEAT and CMA-ES cover a wide area of behaviours, as expected. RL solutions cover smaller areas, with PPO forming distinguishable "worms" in more complicated environments (BipedalWalker), while A2C forms small clouds. This suggests the trust-region restriction of PPO is visible in policy supervectors as well, where the change in behaviour between successive policies is small, whereas A2C updates may change behaviour considerably.

Indeed, we find a positive correlation between total distance travelled by the trained policy and PPO ratio-clip value (Appendix C), which controls the size of the trust region. We also find a negative correlation between distance travelled after an update and average return, indicating that the initial learning steps change the agent's behaviour the most. These results concur with observations of Engstrom et al. [10], where the KL-distance between successive policies first increases and then decreases as training moves on. The results with *behavioural cloning* [40] indicate that distance between the expert policy and trained policy decreases as training progresses, with change to behaviour slowing down as training reaches closer to the expert's performance.

7.5 Applying supervectors to trust-region policy optimization

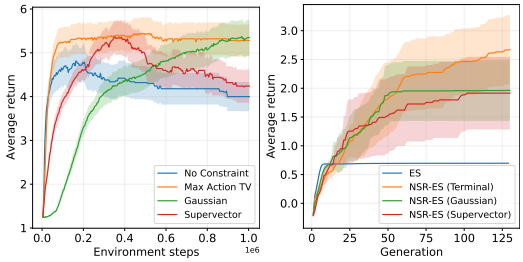
The above insights on trust-region optimization and BC distances suggest that state-based BCs could also be used in trust-region optimization. To test this, we construct a N-dimensional grid world environment where the agent can move to one of N directions per grid, where one direction is correct ($+1$ reward) and rest either reset the episode or do nothing. This environment is constructed to reflect

the doorway scenario: some actions have a large impact, while others do not. Agents are trained with a modified PPO where we disable the policy ratio clip and instead check if state-based BC distance is larger than the threshold after every network update. We compare state-based BCs against no constraint and *total variation divergence* (Max TV) $d(\pi, \pi') = \max_s \frac{1}{2} \sum_a |\pi(a|s) - \pi'(a|s)|$ [45]. We sweep over threshold values and present results for the method with the largest area under the learning curve. Further details are available in Appendix E.

Results in Figure 5a indicate that action-based BC is best suited for this task, but both Gaussian and supervector approaches outperform no constraint. Gaussian BC slowly improves to the highest score overall, while supervector quickly reaches peak and then drops. We believe this is due to the non-gaussian nature of the state distribution, where supervector BC is able to move individual components on different modalities, allowing it to learn fast in the beginning but later become unstable. In the current form, state-based BCs are not practical as part of the training loop, but their advantage lies in the analysis of policies (Section 7.4) and when encouraging novelty (Section 7.6).

7.6 Applying supervectors to novelty search

Novelty search encourages policies to find new behaviours to approach the task [29], which can take the form of maximizing BC distance between policies [8]. While successful, previous work has used domain-specific heuristics like terminal states as BCs to describe the behaviour. Supervectors, or Gaussians, could be used instead as a more generalizable alternative. To test this, we construct a continuous 2D point environment similar to Pacchiano et al. [36], where policy is rewarded for travelling to the positive y-direction. A wall prevents the policy from directly moving in this direction, and the policy has to learn to go around it. We use the methods and code of Conti et al. [8], where novelty encouragement is combined with fitness (NSR-ES) or agent is trained only for fitness (ES). Further details are in Appendix D.



(a) Trust region (b) Novelty search

Figure 5: Results in novelty search (10 repetitions) and trust-region policy optimization (50 repetitions). Shaded region represents 95% confidence interval.

Results in Figure 5b show that both single Gaussian and policy supervector help policy to overcome the obstacle. Using terminal state yields slightly better (but not significantly so) results, likely due to it being an "aligned" [8] BC with the objective, as walking away from previous locations can improve fitness. This demonstrates that supervectors (or even simple Gaussians) could be used as state-based BCs to encourage novel behaviour.

8 Discussion and conclusions

As demonstrated, state-based BCs, and especially policy supervectors based on GMMs, can be used regardless of the environment while also scaling to thousands of policies. Compared to using returns or actions to define behaviour, state-based BC methods are more descriptive and capture dynamics of the environment into a behavioural description. These BCs can be used in the study of policies under different training algorithms, or to encourage exploration/novelty or impose limits on how much behaviour can change per policy update.

While policy supervectors are applicable to studying policies as an outside observer, they are not yet practical as a part of the training loop: they require expensive sampling of the environment (as opposed to reusing already collected samples). This direction could be further explored by creating more sample efficient solutions or differentiable state-based BCs to train agents. Other topics include combining supervectors with the behavioural embeddings framework [36], extending the novelty search experiments [37] and further analysis of parameters learned by policy supervectors, such as the meaning of each adapted Gaussian component.

References

[1] R. Agarwal, M. C. Machado, P. S. Castro, and M. G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021.

[2] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*, pages 1471–1479, 2016.

[3] G. Berseth, D. Geng, C. Devin, C. Finn, D. Jayaraman, and S. Levine. SMiRL: Surprise minimizing rl in dynamic environments. *arXiv preprint arXiv:1912.05510*, 2019.

[4] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

[7] W. M. Campbell, D. E. Sturim, and D. A. Reynolds. Support vector machines using gmm supervectors for speaker verification. *IEEE signal processing letters*, 13(5):308–311, 2006.

[8] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in neural information processing systems*, pages 5027–5038, 2018.

[9] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[10] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020.

[11] B. Everitt and A. Skrondal. *The Cambridge dictionary of statistics*, volume 106. Cambridge University Press Cambridge, 2002.

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

[13] A. Gaier and D. Ha. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, pages 5364–5378, 2019.

[14] A. Gaier and D. Ha. Weight agnostic neural networks. *arXiv preprint arXiv:1906.04358*, 2019.

[15] J.-L. Gauvain and C.-H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE transactions on speech and audio processing*, 2(2):291–298, 1994.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

[17] A. Grover, M. Al-Shedivat, J. Gupta, Y. Burda, and H. Edwards. Learning policy representations in multiagent systems. In *International conference on machine learning*, pages 1802–1811. PMLR, 2018.

[18] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[19] J. Harb, T. Schaul, D. Precup, and P.-L. Bacon. Policy evaluation networks. *arXiv preprint arXiv:2002.11833*, 2020.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[21] D. Hernandez, K. Denamganai, S. Devlin, S. Samothrakis, and J. A. Walker. A comparison of self-play algorithms under a generalized framework. In *IEEE Conference on Games*, 2019.

[22] J. R. Hershey and P. A. Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

[23] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[24] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[25] S. M. Jordan, Y. Chandak, D. Cohen, M. Zhang, and P. S. Thomas. Evaluating the performance of reinforcement learning algorithms. In *International Conference on Machine Learning*, 2020.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[27] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[28] T. Kinnunen and H. Li. An overview of text-independent speaker recognition: From features to supervectors. *Speech communication*, 2010.

[29] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[30] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[31] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[32] B. Matusch, J. Ba, and D. Hafner. Evaluating agents without rewards. *arXiv preprint arXiv:2012.11538*, 2020.

[33] G. J. McLachlan and K. E. Basford. *Mixture models: Inference and applications to clustering*, volume 38. M. Dekker New York, 1988.

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[35] T. Ni, H. Sikchi, Y. Wang, T. Gupta, L. Lee, and B. Eysenbach. f-IRL: Inverse reinforcement learning via state marginal matching. In *Conference on Robot Learning*, 2020.

[36] A. Pacchiano, J. Parker-Holder, Y. Tang, K. Choromanski, A. Choromanska, and M. Jordan. Learning to score behaviors for guided policy optimization. In *International Conference on Machine Learning*, pages 7445–7454. PMLR, 2020.

[37] J. Parker-Holder, A. Pacchiano, K. Choromanski, and S. Roberts. Effective diversity in population-based reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[39] S. Pitis, H. Chan, S. Zhao, B. Stadie, and J. Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, 2020.

[40] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[41] A. Raffin. Rl baselines zoo. https://github.com/araffin/rl-baselines-zoo, 2018.

[42] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[43] R. Raileanu, M. Goldstein, A. Szlam, and R. Fergus. Fast adaptation to new environments via policy-dynamics value functions. In *International Conference on Machine Learning*, pages 7920–7931. PMLR, 2020.

[44] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1-3):19–41, 2000.

[45] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[47] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[48] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[49] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[50] H. Tang, Z. Meng, J. Hao, C. Chen, D. Graves, D. Li, W. Liu, and Y. Yang. Represent your own policies: Reinforcement learning with policy-extended value function approximator. *arXiv preprint arXiv:2010.09536*, 2020.

[51] S. Wang, S. Toyer, A. Gleave, and S. Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. https://github.com/HumanCompatibleAI/imitation, 2020.

[52] Z. Wang, J. S. Merel, S. E. Reed, N. de Freitas, G. Wayne, and N. Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329, 2017.