Supplementary materials for Discovering symbolic policies with deep reinforcement learning

Mikel Landajuela [*1] **Brenden K. Petersen** [*1] **Sookyung Kim** [*1] **Claudio P. Santiago** [1] **Ruben Glatt** [1] **T. Nathan Mundhenk** [1] **Jacob F. Pettit** [1] **Daniel M. Faissol** [1]

1. Pseudocode for Deep Symbolic Policy

Algorithm 1 describes pseudocode for deep symbolic policy with "anchoring" for multi-action environments, trained using the risk-seeking policy gradient with hierarchical entropy regularizer. Algorithm 2 describes the Policy Evaluator, used to compute the reward for a symbolic policy. In both algorithms, the function *Instantiate* generates an executable function $f : \mathcal{S} \to \mathbb{R}$ from the sequence of tokens τ sampled by the Policy Generator. Note that Algorithm 1 references Algorithm 2 by generating a *closure* for the reward function. This captures the previously learned symbolic expressions and the pre-trained "anchor" policy into the reward function.

2. Solution of Uniform Arity Prior

The goal of this section is to find a prior as a logit vector

$$\psi_\circ = \left(\psi_\circ^{1,2}, \dots, \psi_\circ^{n_2,2}, \psi_\circ^{1,1}, \dots, \psi_\circ^{n_1,1}, \psi_\circ^{1,0}, \dots, \psi_\circ^{n_0,0}\right)$$

such that

$$\text{softmax}(\psi_\circ) = \left(\frac{1}{3n_2}\right)_{n_2} \| \left(\frac{1}{3n_1}\right)_{n_1} \| \left(\frac{1}{3n_0}\right)_{n_0}.$$

Using the definition of the softmax operator:

$$\begin{cases} \dfrac{\exp(\psi_\circ^{i,2})}{\sum_{i,j}\exp(\psi_\circ^{i,j})} = \dfrac{1}{3n_2}, \ \forall\, i \in \{1, \dots, n_2\}, \\ \dfrac{\exp(\psi_\circ^{i,1})}{\sum_{i,j}\exp(\psi_\circ^{i,j})} = \dfrac{1}{3n_1}, \ \forall\, i \in \{1, \dots, n_1\}, \\ \dfrac{\exp(\psi_\circ^{i,0})}{\sum_{i,j}\exp(\psi_\circ^{i,j})} = \dfrac{1}{3n_0}, \ \forall\, i \in \{1, \dots, n_0\}. \end{cases}$$

Considering the change of variables $\exp(\psi_\circ^{i,j}) \to x^{i,j}$ in the previous system, we obtain a homogeneous system of linear equations with matrix

$$\mathbf{A} = \mathbf{I}_{n_2+n_1+n_0} - \frac{1}{3} \underbrace{\begin{pmatrix} 1/n_2 & \cdots & 1/n_2 \\ \vdots & & \vdots \\ 1/n_2 & \cdots & 1/n_2 \\ \hline 1/n_1 & \cdots & 1/n_1 \\ \vdots & & \vdots \\ 1/n_1 & \cdots & 1/n_1 \\ \hline 1/n_0 & \cdots & 1/n_0 \\ \vdots & & \vdots \\ 1/n_0 & \cdots & 1/n_0 \end{pmatrix}}_{n_2+n_1+n_0} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} n_2 \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} n_1 \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} n_0 \end{matrix},$$

where $\mathbf{I}_m$ is the identity matrix in $\mathbb{R}^{m\times m}$. We have $\det(\mathbf{A}) = 0$ and $\text{rank}(\mathbf{A}) = n_0 + n_1 + n_2 - 1$. [1] Thus, the system is overdetermined and the solution has one parameter. Fixing

$$x^{1,0} = \frac{\exp(c)}{n_0},$$

with $c \in \mathbb{R}$, the problem becomes well-defined, with solution

$$x^{i,j} = \frac{\exp(c)}{n_j} \ \ \forall\, i \in \{1, \dots, n_j\} \text{ and } \forall\, j \in \{0, 1, 2\}.$$

Inverting the change of variables we obtain

$$\psi_\circ = (-\log n_2)_{n_2} \| (-\log n_1)_{n_1} \| (-\log n_0)_{n_0} + c.$$

3. Expected Initial Expression Length

We informally show that the expected initial expression length under the uniform arity prior ($\psi_\circ$) is infinite. We assume that the RNN emissions are all zero (as in the start of training) and there are no constraints or priors besides $\psi_\circ$. That is, $p(\tau_i = \mathcal{L}_j) = \text{softmax}(\psi_\circ)_j$.

When generating an expression as a sequence of tokens, we can track a "counter" δ for the number of remaining nodes

[*]Equal contribution [1]Lawrence Livermore National Laboratory, Livermore, California, USA. Correspondence to: Brenden K. Petersen <bp@llnl.gov>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]We do not provide a proof of these properties of $\mathbf{A}$ in general, but one can easily check that they hold for all relevant values of n_0, n_1 and n_2 considered in DSP.

Algorithm 1 Deep symbolic policy with "anchoring" for multi-action environments

Function DeepSymbolicPolicy($\alpha, \epsilon, \eta, \gamma, \mathcal{E}, \Psi, M, N$)
Input: Learning rate α, risk factor ϵ, entropy weight η, entropy decay γ, environment $\mathcal{E}$ with observation space $\mathcal{S} \subset \mathbb{R}^m$ and action space $\mathcal{A} \subset \mathbb{R}^n$, pre-trained "anchor" policy Ψ (if $n > 1$), maximum number of iterations M, number of training episodes N
Output: Fully symbolic policy f
for *action dimension* $i = 1, \ldots, n$ **do**
  Initialize RNN with parameters θ, defining distribution over expressions $p(\cdot|\theta)$
  $R(\cdot) \leftarrow$ PolicyEvaluator$(\cdot, \bar{f}_1, \ldots, \bar{f}_{i-1}, \Psi, \mathcal{E}, N)$ `// Define closure for Policy Evaluator`
  $\tau^\star \leftarrow$ null `// Initialize best expression`
  for *iteration* $= 1, \ldots, M$ **do**
    $\mathcal{T} = \{\tau^{(i)} \sim p(\cdot|\theta)\}_{i=1,\ldots,N}$ `// Sample expressions via Policy Generator`
    $\mathcal{R} = \{R(\tau^{(i)})\}_{i=1,\ldots,N}$ `// Compute rewards`
    $R_\epsilon = (1-\epsilon)$ percentile of $\mathcal{R}$ `// Compute reward threshold`
    $\mathcal{T}_\epsilon = \{\tau^{(i)} : R(\tau) \geq R_\epsilon\}$ `// Select subset of expressions above threshold`
    $\hat{g}_1 = \frac{1}{|\mathcal{T}_\epsilon|} \sum_{\tau \in \mathcal{T}_\epsilon} ((R(\tau) - R_\epsilon) \nabla_\theta \log p(\tau|\theta))$ `// Risk-seeking policy gradient`
    $\hat{g}_2 = \frac{1}{|\mathcal{T}_\epsilon|} \sum_{\tau \in \mathcal{T}_\epsilon} \left(\eta \sum_{i=1}^{|\tau|} \gamma^{i-1} \nabla_\theta H[p(\tau_i|\tau_{1:(i-1)}; \theta)] \right)$ `// Hierarchical entropy gradient`
    $\theta \leftarrow \theta + \alpha(\hat{g}_1 + \hat{g}_2)$ `// Apply gradients`
    if $\max \mathcal{R} > R(\tau^\star)$ **then** $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ `// Update best expression`
  end
  $\bar{f}_i \leftarrow$ Instantiate$(\tau^\star)$ `// Set fixed sub-policy for next action dimension`
end
$f \leftarrow \langle \bar{f}_1, \ldots \bar{f}_n \rangle$ `// Final policy is fully symbolic`
return f

Algorithm 2 Policy Evaluator, used to compute reward for symbolic policy τ

Function PolicyEvaluator($\tau, \bar{f}_1, \ldots, \bar{f}_{i-1}, \Psi, \mathcal{E}, N$)
Input: Symbolic policy being evaluated τ, previously learned fixed symbolic expressions $\bar{f}_1, \ldots, \bar{f}_{i-1}$, pre-trained "anchor" policy Ψ, environment $\mathcal{E}$, number of training episodes N
Output: Reward R for symbolic policy τ
$R \leftarrow 0$
$f \leftarrow$ Instantiate(τ)
for *episode* $= 1, \ldots, N$ **do**
  $s \leftarrow$ Reset$(\mathcal{E})$ `// Sample new starting state`
  while $\mathcal{E}$ *is non-terminal* **do**
    $a \leftarrow \langle \bar{f}_1(s), \ldots, \bar{f}_{i-1}(s), f(s), \Psi_{i+1}(s), \ldots, \Psi_n(s) \rangle$ `// Compute action`
    $s, r \leftarrow$ Execute$(\mathcal{E}, a)$ `// Step the environment`
    $R \leftarrow R + r$
  end
end
$R \leftarrow \frac{R}{N}$ `// Average episodic rewards`
return R

in the expression tree. When selecting a binary token, δ is incremented. When selecting a unary token, δ does not change. When selecting a terminal token, δ is decremented. Before tokens are sampled, δ begins at 1. The expression completes when $\delta = 0$, at which point all branches reach terminal nodes. For example, the expression $\sin(x)(x+1)$ can be represented using the sequence $[\times, \sin, x, +, x, 1]$ which has a corresponding counter sequence $\delta = [2, 2, 1, 2, 1, 0]$ and expression length of 6 tokens.

Using the above knowledge, we can reformulate the expression length as a one-dimensional random walk that begins at state $s_0 = 1$ and ends at state $s_T = 0$. In general, let t_n be the expected number of steps to end a random walk beginning at $s_0 = n$ and ending at $s_T = 0$. Let p be the probability of incrementing s (analogous to selecting a binary token), q be the probability of decrementing s (analogous to selecting a terminal token), and r be the probability that s is unchanged (analogous to selecting a unary token). Consider the first step: starting from $s_0 = 1$, there are three possible values for s_1. With probability p, $s_1 = 2$ and the expected number of remaining steps becomes t_2. With probability q, $s_1 = 0$ and thus the walk ends. With probability r, $s_1 = 1$ and the expected number of remaining steps remains t_1. Thus, using $p + q + r = 1$, it follows

$$\begin{aligned} t_1 &= p \cdot (1 + t_2) + q \cdot (1) + r \cdot (1 + t_1) \\ &= 1 + pt_2 + rt_1 \text{ with } t_1 \in [0, +\infty). \end{aligned}$$

By linearity of expected value, we have $t_n = t_1 + t_{n-1} = t_1 + t_1 + t_{n-2} = \cdots = nt_1$. Thus, we can write:

$$\begin{aligned} t_1 &= 1 + (2p + r)t_1 \\ &= 1 + (1 + p - q)t_1 \text{ with } t_1 \in [0, +\infty). \end{aligned}$$

There are two cases. If $p < q$, then $t_1 = \frac{1}{q-p}$. If $p \geq q$, then $t_1 = +\infty$. That is, the expected expression length is infinite when the probability of a binary token is greater than or equal to the probability of a terminal token. Under the uniform arity prior, we have $p_2 = p_1 = p_0 = \frac{1}{3}$ and thus the expected length is infinite.

4. Additional Results and Discussion

Heatmaps of discovered symbolic policies. The dramatic reduction in the complexity of the symbolic policies discovered by DSP is illustrated in Figure 4 for each action dimension and for each environment. For environments with observation space $\mathcal{S} \subset \mathbb{R}^m$ greater than two dimensions ($m > 2$), we report 2-D plots by considering "slices" of the space such that the x- and y-axes represent the first and second "half" of the observation space, respectively. In particular, the x-axis represents the slice such that $\tilde{s}_1 = \cdots = \tilde{s}_{\lfloor \frac{m}{2} \rfloor}$, and the y-axis represents the slice such that $\tilde{s}_{\lfloor \frac{m}{2} \rfloor + 1} = \cdots = \tilde{s}_m$, where $\tilde{s}_i$ is the ith dimension of the normalized observation space. Each dimension of the observation space is normalized to [0, 1] using the empirical low and high values of s_i, based on the dataset used for the the regression baseline. Color values in heatmaps represent the normalized action values. There is a stark contrast in the complexity of the policies discovered by DSP and the neural network-based policies trained using DRL.

Intermediate hybrid policies for multi-action environments. For multi-action environments, we learn the symbolic actions one at a time, distilling the best NN-based policy available. In this section we report the scores of the intermediate hybrid policies, i.e. those including symbolic and NN-based actions, that are built during training. For Lunarlander, the hybrid policy $\{a_1, \Psi(s)_2\}$, where a_1 is given in Table 1 and $\Psi(s)$ is the corresponding anchor, obtained a score of 278.41. Note that this policy improves upon the anchor, which obtained a score of 272.65. On the other hand, the hybrid policies $\{a_1, \Psi(s)_2, \Psi(s)_3\}$ and $\{a_1, a_2, \Psi(s)_3\}$ for Hopper, obtained scores of 2494.29 and 2299.91, respectively. For reference, the score of the anchor reported in Table 2 was 2741.86. For BipedalWalker, the hybrid policies $\{a_1, \Psi(s)_2, \Psi(s)_3, \Psi(s)_4\}$, $\{a_1, a_2, \Psi(s)_3, \Psi(s)_4\}$ and $\{a_1, a_2, a_3, \Psi(s)_4\}$, obtained scores of 160.57, 181.34, and 272.35, respectively.

Stability analysis for MountainCar. The discrete dynamics of the MountainCar system are given by:

$$\begin{aligned} v(t_{k+1}) &= 0.0015a_1(t_k) + v(t_k) - 0.0025\cos(3x(t_k)), \\ x(t_{k+1}) &= v(t_{k+1}) + x(t_k), \end{aligned}$$

where $x(t_k)$ is position, $v(t_k)$ is velocity, and $a_1(t_k)$ is the action at time step $t_k = t_0 + k\Delta t$ and $k \in \mathbb{N}$. Linearization of the *uncontrolled* system around the perturbed equilibrium state $s_{\text{eq}} = (x, v) = (-0.52, 0)$ (the bottom of the valley) yields a Jacobian matrix with eigenvalues $(0.99625 + 0.0865213\, i, 0.99625 - 0.0865213\, i)$. The norm of these eigenvalues is exactly 1.0, and thus, any solution of the uncontrolled system starting in a neighborhood of s_{eq} oscillates around s_{eq} indefinitely (see Brunton and Kutz (2019)). Substituting $a_1(k)$ with the policy discovered by DSP yields eigenvalues $(1.05068 + 0.0702259\, i, 1.05068 - 0.0702259\, i)$. These have norm 1.05302, showing that any solutions starting close to s_{eq} moves away from s_{eq}. That is, the discovered policy makes the attractor point s_{eq} function as a repeller, and the car is successfully pushed to the goal. This phenomenon is observable in Fig 5b.

Stability analysis for Pendulum. The continuous dynamics of the Pendulum system are defined by :

$$\theta''(t) = \frac{3a_1(t)}{l^2 m} + \frac{3g\sin(\theta(t))}{2l},$$

where g is gravitational acceleration, m is the pendulum mass, l is the pendulum length, $\theta(t)$ is the angle with respect

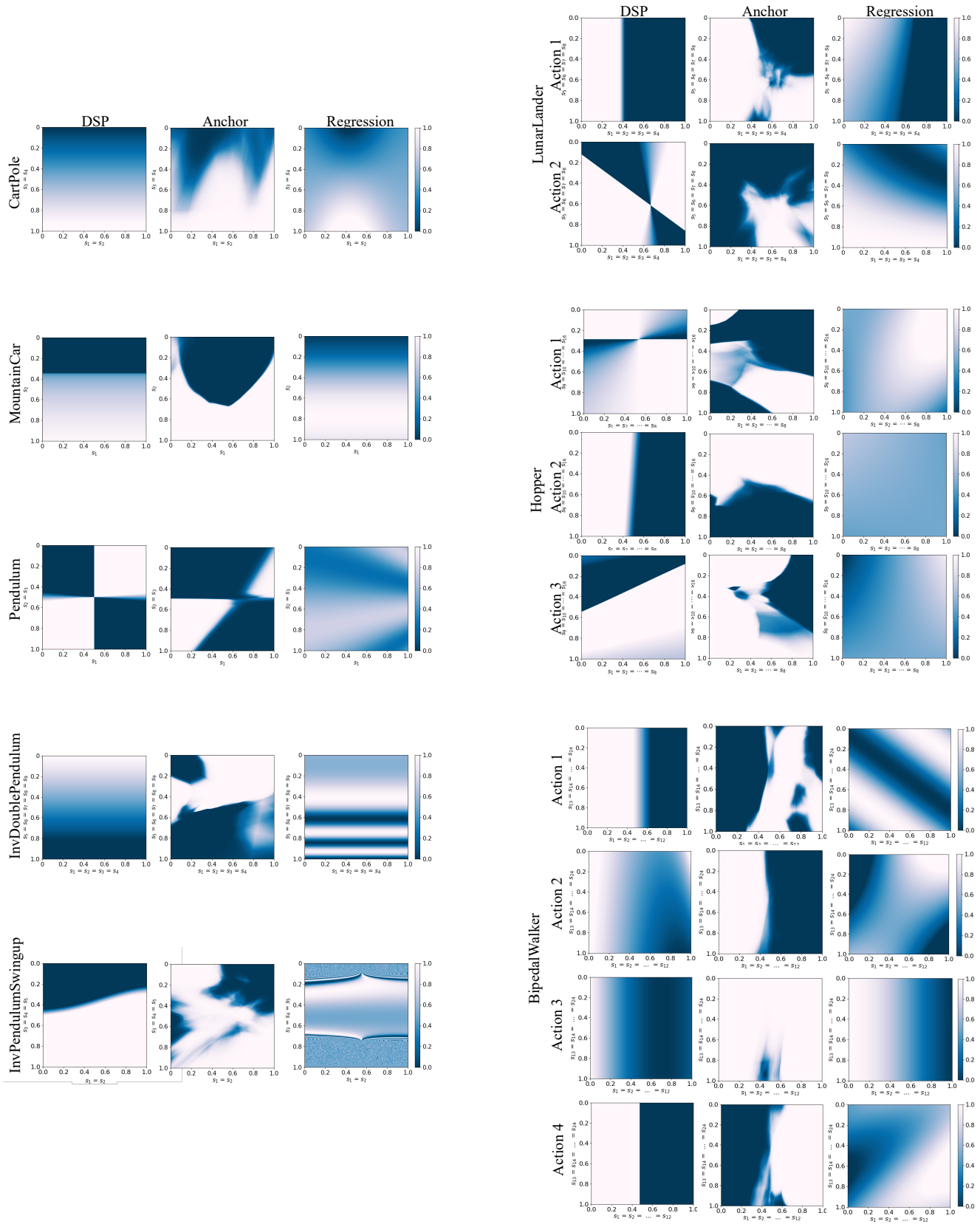

Figure 4: Heatmaps of actions computed from the symbolic policies discovered by DSP (left columns), the anchor model (middle columns), and the symbolic regression baseline (right columns) for all environments. [Left] Five single-action environments. [Right] Two multi-action environments. For environments with more than two observation dimensions, we plot slices of the observation space (detailed in the text). Each axis is normalized to $[0, 1]$ using empirical observation bounds. Color values of heatmap represent action values, normalized by the action bounds to $[0, 1]$.

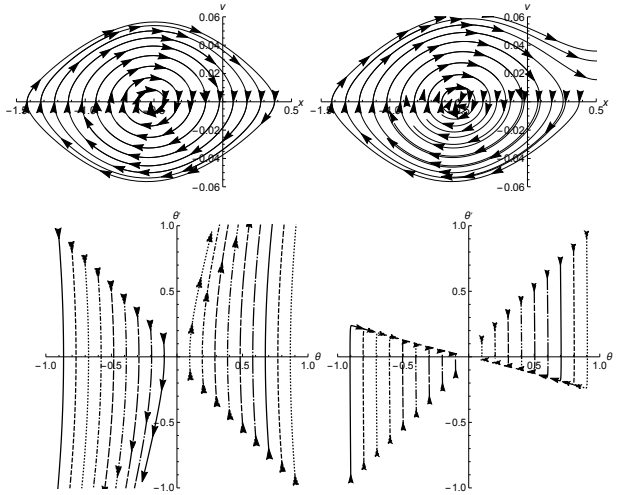

Figure 5: Phase portraits of the uncontrolled (left) and controlled (right) systems in MountainCar (top) and Pendulum (bottom).

the vertical axis, and $a_1(t)$ is the action. Linearization of the uncontrolled system around the equilibrium point $s_{\text{eq}} = (\theta, \theta') = (0, 0)$ yields a Jacobian matrix with eigenvalues $(-3.87, 3.87)$. Since the second eigenvalue is positive, s_{eq} is an unstable equilibrium according to the Hartman–Grobman theorem. Substituting the control $a_1(t)$ found by DSP in the system gives eigenvalues $(-59.7489, -0.25105)$. Since all eigenvalues are non-positive, the symbolic policy discovered by DSP controls the Pendulum around s_{eq}. Again, the phase portraits in Fig 5c show the symbolic policy controlling the system across a wide range of starting conditions.

5. Interpreting Learned Symbolic Policies

We provide a simple interpretation for the symbolic policies discovered by DSP from Table 1. We exclude an interpretation for the Hopper environment because the meanings of each observation and action dimension are not described in the documentation or code.

CartPole. The goal is to prevent a pole on a cart from falling over. The cart is controlled by a continuous force that pushes the cart to the right $(+)$ or left $(-)$. The discovered policy consists of two terms, involving the pole angle s_3 and the pole velocity s_4. The first term pushes the cart in whichever direction the pendulum is currently leaning, which provides a stabilizing effect. The second term smoothes these dynamics by taking into account how fast the pendulum is falling. Interestingly, the policy discovered by DSP has structure $a_1 = \boldsymbol{K}\boldsymbol{s}$, where $\boldsymbol{s}$ is the state and $\boldsymbol{K}$ is a gain vector. This functional form is closely related to the typical feedback controllers that are ubiquitous in linear control theory.

MountainCar. The goal is to make an under-powered car escape from the bottom of a one-dimensional valley using minimal effort. The geometry of the valley is a fixed concave shape. States consist of the car position (s_1) and its velocity (s_2). The action is the force to push the car, either to the left $(-)$ or to the right $(+)$. The policy discovered by DSP (in Table 1) only involves s_2. Recall that the operator $\log$ is protected, such that that $\log(s_2) = 0$ for $s_2 < 0$. Thus, the control is totally switched off whenever the car is moving left. For $s_2 \geq 0$, the force is positive, monotonic, and increasing. Thus, the policy pushes the car whenever it is moving to the right, and the force magnitude is maximum when the car is in the bottom of the valley. In other words, the policy builds momentum in the car, letting it swing back and forth through the valley (see Supplementary Video). By letting the car move freely on the backswing, it drastically reduces the energy injected into the system, thus achieving near-perfect reward.

Pendulum. The goal is to keep a frictionless pendulum standing upward by taking actions to move its joint rightward $(+)$ or leftward $(-)$. States consist of $\cos(\theta)$ (s_1), $\sin(\theta)$ (s_2), where θ is the angle of the pendulum, and its angular velocity ω (s_3). The environment incurs a cost of the form $\sum(\theta^2(t) + 0.1\omega^2(t) + 0.001a_1(t))$, i.e, the goal is to keep at zero angle (vertical), with the least rotational velocity and effort. The negative constant multiplying s_2 and s_3 in the discovered policy prescribes a smooth torque in the opposite direction of the angle in which the pendulum is currently leaning. The policy switches off when in the goal state $(s_1 = 1, s_2 = 0, s_3 = 0)$.

InvertedDoublePendulum. The goal and action is the same as CartPole, but the system has an additional pole with additional states. The discovered policy is a function of $s_8 = \sin(\gamma)$ exclusively, where γ is the angle of the joint connecting the two poles. Thus, the discovered policy switches the direction and magnitude of the force acting on the cart according to γ. This is a very parsimonious strategy that obtains a reasonable performance. As in CartPole, the discovered symbolic policy has the structure of a classical feedback controller in linear control theory.

InvertedPendulumSwingup. This task is a combination of CartPole and Pendulum, with the same goal as Pendulum. Actions move the cart to the left $(-)$ or right $(+)$. We can identify two components in the discovered policy. First, we have a polynomial function involving s_1 (location of cart), s_4 ($\sin(\theta)$, where θ is the angle between pole and cart), and s_5 (angular velocity). This first contribution has the structure of a linear feedback controller. The second part is a non-linear function involving sinusoidal functions of simple polynomials over the state variables. This second term has bounded range within $[-2, 2]$ and contributes to the core linear feedback controller providing additional control

Table 3: Best discovered symbolic policies using the regression baseline.

Environment	Best variant	Expression
CartPole	Clipped	$a_1 = s_4 \exp(-\sin(s_1 + s_2))$
MountainCar	Unclipped	$a_1 = \sin(5.0 s_2 (9.0 - s_2))$
Pendulum	Clipped	$a_1 = \sin(0.2 s_3 \exp(s_1) + 0.2 s_3 + 0.2)$
InvDoublePend	Unclipped	$a_1 = -\sin(2.9 s_9 + (s_5 + s_9)(10.0 s_8 + s_9 + \exp(s_5 s_9)))$
InvPendSwingup	Clipped	$a_1 = \sin(s_3 s_4(-s_1 \exp(\exp(s_4 s_5) - 0.37 \exp(-s_1 + s_2 + s_4)) + 5.0))$
LunarLander	Clipped	$a_1 = -s_2 - s_4 \exp(s_3) \exp(\exp(s_4)) - \sin(s_7) + 0.1$ $a_2 = \sin(s_1 s_5 - s_1 + 2 s_5)$
Hopper	Unclipped	$a_1 = s_4 \sin(2 s_{15} + s_4)$ $a_2 = s_8(s_{11} - s_{15})$ $a_3 = s_1(-s_{14} + 2 s_{15})$
BipedalWalker	Unclipped	$a_1 = \sin(s_{13} + s_{14} - s_5 - 2 s_7)$ $a_2 = s_{10}(-s_{11} - s_{13} - s_{22} + s_5)$ $a_3 = -\sin(3 s_{10} + s_{12} - s_3 - s_7)$ $a_4 = s_{10} + s_{20} \sin(s_{13} + s_6)$

in the edge cases.

LunarLander. The goal is to land a spacecraft on a landing pad without crashing. There are two actions: a_1 controls the main engine and a_2 controls the side thrusters. The discovered symbolic policy for a_1 is mainly dependent on s_2 (the height) and s_4 (the vertical component of velocity). The main engine turns on to counteract downward motion, eventually shutting off when the spacecraft is safe to free-fall without crashing. The discovered symbolic policy for a_2 is a function of s_4, s_6 (angular velocity), and s_3 (horizontal component of velocity). The dominant term determining the sign of a_2 is $s_6 - s_3$, deciding which part of the engine should be fired between left or right based on the tilting tendency of the spacecraft effectively moving it towards the center of the landing pad.

BipedalWalker. In this task, the agent has to learn to move a bipedal walker forward, with the goal of reaching the far end of a plane with uneven terrain. The state consists of 24 values: hull angle s_1, angular hull velocity s_2, normalized horizontal velocity s_3, normalize vertical velocity s_4, first leg hip joint angle s_5, first leg hip joint torque s_6, first leg knee joint angle s_7, first leg knee joint torque s_8, first leg ground contact s_9, second leg hip joint angle s_{10}, second leg hip joint torque s_{11}, second leg knee joint angle s_{12}, second leg knee joint torque s_{13}, second leg ground contact s_{14}, and ten lidar range finder measurements s_{15} to s_{24}. There are four actions: first leg hip motor torque a_1, first leg knee motor torque a_2, second leg hip motor torque a_3, and second leg knee motor torque a_4. We first note that the lidar measurements have very little influence on the symbolic policies: only one of the ten measurements appears in the expression for action a_1, and another in a_2. We also see that each action heavily depends on the state variables of the respective joint, augmented by information about the hull or other joints. Actions a_1 and a_3 present a linear or quasi-linear structure. Actions a_2 and a_4 are highly non-linear, suggesting that they require more complex and subtle responses to the state of the system. Remarkably, action a_4 depends only on s_2 and (linearly) on s_{10}. Furthermore, we see that $a_4 < 0$ when $s_2 s_{10} < 0$, and $a_4 > 0$ when $s_2 s_{10} > 0$, which points to a non-trivial relationship between the angular hull velocity and the second leg hip joint angle that controls the sign of the second leg knee motor torque.

6. Additional Experiment Details

Regression baseline. The 'regression" baseline involves performing symbolic regression on an offline dataset generated from the top-performing pre-trained Zoo policy for each environment. Consider an environment with observation space $\mathcal{S} \subset \mathbb{R}^m$ and action space $\mathcal{A} \subset \mathbb{R}^n$. To generate the dataset for this environment, we first select the top-performing pre-trained Zoo policy. We then execute this neural network-based policy in the environment for 1,000 episodes, storing each tuple $(s_1, \ldots, s_m, a_1, \ldots, a_n)$ as a datapoint. After collecting all datapoints, we uniform randomly select 10,000 points to be used as the dataset for symbolic regression.

We use deep symbolic regression (Petersen et al., 2021) to perform symbolic regression. Since this algorithm is stochastic, we repeat each experiment 10 times using different random seeds. We select the expression with the lowest

mean-square error on the held-out test data (using 80%-20% train-test split) to be evaluated in the environment. Unlike DSP, in which each dimension of the action space must be learned jointly, symbolic regression can be performed independently for each action dimension. Thus, one experiment consists of running n iterations of symbolic regression to learn symbolic representations of actions $a_1, \ldots, a_n$. Specifically, the ith iteration of symbolic regression learns the function $f : \mathcal{S} \to \mathbb{R}$, using the above dataset with $s_1, \ldots, s_m$ as the predictors and a_i is the outcome variable (ignoring $a_{j \neq i}$).

When executing symbolic policies in the environment, action values are clipped to fall within the environment's action space, which is typically bounded. (Even if we do not clip actions ourselves, environments typically do this internally.) However, clipping only *after* symbolic regression has been performed may limit symbolic regression from learning policies that exploit these bounds (e.g. a "bang-bang" policy that quickly switches from one extreme of the action space to the other). Thus, we performed an additional variant of the symbolic regression experiments in which the generated functions f are first clipped to the bounds of the action space before being compared to the data for the regression fit. Thus, expressions are not penalized for large errors outside the bounds of the action space. (Note that the offline dataset need not be clipped, as all DRL algorithms used to generate pre-trained policies had their own mechanism for ensuring predicted actions fall within the bounds of the action space.) Using this variant, expressions generated by symbolic regression are essentially altered as:

$$f'(s) = \begin{cases} \mathcal{A}_i^{\text{high}} & f(s) \geq \mathcal{A}_i^{\text{high}} \\ f(s) & \mathcal{A}_i^{\text{low}} < f(s) < \mathcal{A}_i^{\text{high}} \\ \mathcal{A}_i^{\text{low}} & f(s) \leq \mathcal{A}_i^{\text{low}} \end{cases},$$

where $\mathcal{A}_i^{\text{low}}$ and $\mathcal{A}_i^{\text{high}}$ are the low and high bounds of the ith dimension of the action space, respectively. An example where this clipping may be desirable is illustrated in Fig 6.

For each environment, we selected the variant with the best environment score. Table 3 shows the best symbolic policy obtained from the regression baseline, along with which variant (*Clipped* or *Unclipped*) performed best. These expressions correspond to the scores reported in Table 2.

Constant optimization. Recall that the objective function for constant optimization is:

$$\gamma^\star = \underset{\gamma \in D(\gamma^0, \underline{r}, \overline{r})}{\arg\max} \; \mathbb{E}\left[R(\tau; \gamma)\right].$$

Each algorithm used for constant optimization includes the following parameters:

- γ^0: initial solution

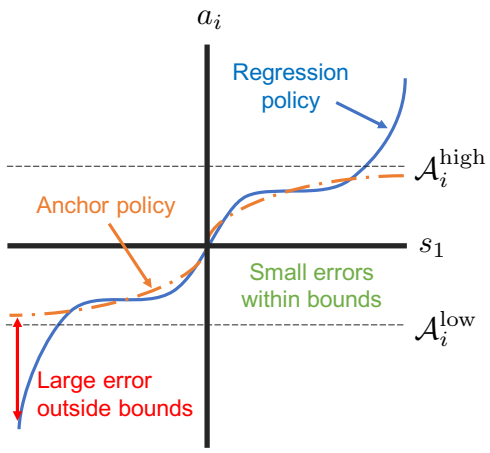

Figure 6: Example of problem in regression baseline without clipping actions. There can be large regression errors when the action values from regression fall far outside the bounds of the action space. Clipping the predicted actions would prevent the regression baseline from being penalized outside these bounds.

- ϵ, parameter used to construct the bounds: $\{0.5, 1.0\}$
- the number of episodes used to estimate the objective function: $\{100, 200\}$ (for single-action environments) and $\{10, 20\}$ (for multi-action environments)
- number of restart points: $\{100, 200\}$ (single-action) and $\{10, 20\}$ (multi-action)

For algorithms that consider bounds, we ran the optimization both with and without bounds. The factors $\underline{r}$ and $\overline{r}$, used to determine the bounds $\underline{r}\gamma^0$ and $\overline{r}\gamma^0$, are computed using ϵ for the i-th coordinate:

$$\underline{r} = \frac{\min(\gamma_i^0 - \epsilon\gamma_i^0, -1)}{\gamma_i^0},$$
$$\overline{r} = \frac{\max(\gamma_i^0 + \epsilon\gamma_i^0, 1)}{\gamma_i^0}.$$

The minimum and maximum operators ensure we provide each coordinate with a minimum significant variation.

The restart points are generated coordinate-wise using a truncated normal distribution with mean γ^0 and standard deviation $\epsilon^2(\overline{r} - \underline{r})\gamma^0$, bounded by D. They are generated in the same way for both bounded and unbounded optimization. Thus, for unbounded optimization experiments, bounds are computed only to calculate the parameters of the distribution from which restart points are drawn.

We use the following optimization methods: Trust Region (bounded) (Byrd et al., 1987), Nelder-Mead (unbounded) (Nelder and Mead, 1965), Constrained Optimization by Linear Approximation (unbounded) (Powell, 1995), Sequential

Least Squares (bounded) (Carayannis et al., 1983), Broyden-Fletcher-Goldfarb-Shanno (bounded and unbounded) (Head and Zerner, 1985), and Bayesian Optimization (bounded) (Nogueira, 2014). For each action dimension in each environment, we select the algorithm that yielded the highest average episodic reward across 1,000 evaluation episodes. For Bayesian optimization, we used the implementation in Nogueira (2014). For all other algorithms, we used implementations in SciPy (Virtanen et al., 2020).

Training details. Since executing the environment is the computational bottleneck, DSP was run until a maximum number of N_{total} environment episodes were executed. We used $N_{\text{total}} = 2\text{M}$ episodes for single-action environments and $N_{\text{total}} = 400,000$ episodes for multi-action environments (which tend to be more computationally expensive). The number of DSP training steps depends on both batch size and N_{train} via: N_{total} = number of training steps $\times$ batch size $\times$ N_{train}.

We introduced a Boolean hyperparameter that controls whether to fix the N_{train} environment seeds for each reward computation. If `True`, this has the benefit of rendering the task deterministic; that is, repeated computations of the reward using the same symbolic policy will yield identical rewards. However, it also introduces the risk of the symbolic policy to overfit to that set of environment seeds. Further, if the set of N_{train} fixed seeds is a poor representation of the starting state distribution, evaluation performance may be degraded. If `False`, these generalization risks are avoided but the reward becomes highly stochastic.

After training completes, we select the symbolic policies with the top 100 rewards and evaluate those using N_{eval} episodes with fixed environment seeds (note there is no overlap between training and evaluation seeds). We report the symbolic policy with the highest evaluation score.

The RNN used was a single-layer LSTM with 32 cells, trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.001. The risk factor used for risk-seeking policy gradients was 0.1. Expression were constrained to a minimun length of 4 and a maximum length of 30.

Hyperparameter selection. Hyperparameters were tuned independently for each action dimension and for each environment. To tune hyperparameters, we performed a small grid search, considering batch size $\in \{100, 200\}$, training episodes per reward evaluation $N_{\text{train}} \in \{5, 10, 20\}$, fixing training seeds $\in \{\texttt{True}, \texttt{False}\}$ entropy weight $\eta \in \{0.01, 0.02\}$, entropy decay $\gamma \in \{0.85, 1.0\}$ and soft length prior $\in \{\texttt{True}, \texttt{False}\}$. No other hyperparameters were tuned. For each hyperparameter combination, we performed 3 independent runs of DSP. We selected the hyperparameter combination with the highest evaluation score (average episodic reward across 1,000 episodes).

Computing infrastructure. Experiments were executed on an Intel Xeon E5-2695 v4 equipped with NVIDIA Tesla P100 GPUs, with 32 cores per node, 2 GPUs per node, and 256 GB RAM per node.

7. Ablation Studies for Exploration Techniques

We consider ablation studies for the two inverted pendulum environments (InvertedDoublePendulum and InvertedPendulumSwingup) using the following variants:

- SE: Standard entropy regularizer $\mathcal{H}$
- HE: Hierarchical entropy regularizer $\mathcal{H}_\gamma$ ($\gamma = 0.85$)
- SLP: Soft length prior ($\lambda = 10, \sigma^2 = 5$) with SE
- SLP+HE: Soft length prior with HE

In Table 4, we report the mean and standard deviation of the evaluation metric (average episodic reward over 1,000 episodes) of the best policy discovered by DSP, averaged over 10 independent training runs. We consider the best hyperparameters found in the hyperparameter study for each environment and consider $N_{\text{total}} = 1\text{M}$. Using the proposed techniques separately (variants HE and SLP), the results improve over the baseline (SE). The best results are obtained when both methods SLP and HE are used together (SLP+HE).

Evaluating DSP solely on the benchmark RL environments is challenging for two reasons: (1) the optimal policy is unknown, precluding a simple "success" criteria; and (2) environments are computationally expensive, preventing large numbers of replicates. To address these issues and provide stronger evidence of the performance gains using our two exploration techniques, we consider the following deterministic environment with one-dimensional action space $\mathcal{A} \subset \mathbb{R}$, m-dimensional state space $\mathcal{S} \subset \mathbb{R}^m$, and a single episodic step $T = 1$. Given a *target policy* $g(\boldsymbol{s})$ and a dataset of states $\{\boldsymbol{s}^{(i)} \in \mathbb{R}^m\}_{i=1}^n$, we define the instantaneous deterministic reward at time step $t = 1$ as

$$r_1 = 1/(1 + \text{NRMSE}_g(\tau)),$$

where

$$\text{NRMSE}_g(\tau) = \frac{1}{\sigma}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(g(\boldsymbol{s}^{(i)}) - f_\tau(\boldsymbol{s}^{(i)}))^2},$$

where $f_\tau(\boldsymbol{s})$ is the expression defined by τ and σ is the standard deviation of the set $\{g(\boldsymbol{s}^{(i)})\}_{i=1}^n$. Note that this environment reduces to the classical problem of symbolic

Table 4: Performance comparison of DSP using SE, HE, and SLP, in the InvertedDoublePendulum and InvertedPendulum-Swingup environments. Values are mean ± standard deviation (10 independent runs) of the episodic reward (averaged over 1,000 episodes) of the best policy discovered by DSP.

Environment	SE	HE	SLP	SLP+HE
InvDoublePend	9062.83 ± 38.68	9132.33 ± 16.21	9105.95 ± 58.35	$\mathbf{9140.32 \pm 6.11}$
InvPendSwingup	861.83 ± 59.16	887.27 ± 1.04	887.09 ± 1.36	$\mathbf{887.28 \pm 1.46}$

Table 5: Recovery rate comparison of DSP using SE, HE, and SLP in the deterministic environments with target policy $g(\boldsymbol{s})$.

Environment with target policy $g(\boldsymbol{s})$		SE	HE	SLP	SLP+HE
$m=1$	$g(s_1) = s_1^3 + s_1^2 + s_1$	100%	100%	100%	100%
$m=1$	$g(s_1) = s_1^4 + s_1^3 + s_1^2 + s_1$	100%	100%	100%	100%
$m=1$	$g(s_1) = s_1^5 + s_1^4 + s_1^3 + s_1^2 + s_1$	100%	100%	100%	100%
$m=1$	$g(s_1) = s_1^6 + s_1^5 + s_1^4 + s_1^3 + s_1^2 + s_1$	100%	99%	100%	100%
$m=1$	$g(s_1) = \sin(s_1^2)\cos(s_1) - 1$	72%	75%	91%	97%
$m=1$	$g(s_1) = \sin(s_1) + \sin(s_1 + s_1^2)$	100%	100%	100%	100%
$m=1$	$g(s_1) = \log(s_1 + 1) + \log(s_1^2 + 1)$	35%	71%	48%	82%
$m=1$	$g(s_1) = \sqrt{s_1}$	96%	97%	100%	100%
$m=2$	$g(s_1, s_2) = \sin(s_1) + \sin(s_2^2)$	100%	100%	100%	100%
$m=2$	$g(s_1, s_2) = 2\sin(s_1)\cos(s_2)$	100%	100%	78%	74%
$m=2$	$g(s_1, s_2) = s_1^{s_2}$	100%	100%	91%	90%
$m=2$	$g(s_1, s_2) = s_1^4 - s_1^3 + \frac{1}{2}s_2^2 - s_2$	0%	0%	0%	0%
	Average	83.6%	86.8%	84.0%	**86.9%**

regression. Thus, we choose functions $g(\boldsymbol{s})$ and datasets of states $\{\boldsymbol{s}^{(i)} \in \mathbb{R}^m\}_{i=1}^n$ according to (Uy et al., 2011), a standard set of benchmark problems for symbolic regression.

By design, this environment has a known global optimal policy; thus, we can measure *recovery rate*, or the fraction of independent training runs in which DSP produces a policy exactly symbolically equivalent to the target policy $g(\boldsymbol{s})$. Since the environment is computationally expedient, we report recovery rate over 100 independent runs, increase the batch size to 1,000 with a risk factor of 0.05, and reduce learning rate to 0.0005, for all variants. In Table 5, we provide the best result obtained with each variant sweeping across $\eta \in \{0.0005, 0.001, 0.01, 0.02\}$ and keeping the rest of parameters fixed. We observe improvements in average recovery rate with both HE and SLP techniques separately. The combination of both provides the best results. Finally, it is worth noting that both contributions improve upon the state-of-the-art results for symbolic regression reported in Petersen et al. (2021).

References

Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *Proc. of the International Conference on Learning Representations*, 2021.

Richard H Byrd, Robert B Schnabel, and Gerald A Shultz. A trust region algorithm for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 24(5): 1152–1170, 1987.

John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

M J D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, 1995.

George Carayannis, D Manolakis, and Nicholas Kalouptsidis. A fast sequential algorithm for least-squares filtering and prediction. *IEEE transactions on acoustics, speech, and signal processing*, 31(6):1394–1402, 1983.

John D Head and Michael C Zerner. A broyden—fletcher—goldfarb—shanno optimization procedure for molecular geometries. *Chemical physics letters*, 122(3):264–270, 1985.

F Nogueira. Bayesian optimization: Open source constrained global optimization tool for python. *URL https://github. com/fmfn/BayesianOptimization*, 2014.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3): 261–272, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.