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]

Abstract

Deep reinforcement learning (DRL) has proven successful for many difficult control problems by learning policies represented by neural networks. However, the complexity of neural network-based policies—involving thousands of composed non-linear operators—can render them problematic to understand, trust, and deploy. In contrast, simple policies comprising short symbolic expressions can facilitate human understanding, while also being transparent and exhibiting predictable behavior. To this end, we propose *deep symbolic policy*, a novel approach to directly search the space of symbolic policies. We use an autoregressive recurrent neural network to generate control policies represented by tractable mathematical expressions, employing a risk-seeking policy gradient to maximize performance of the generated policies. To scale to environments with multi-dimensional action spaces, we propose an "anchoring" algorithm that distills pre-trained neural network-based policies into fully symbolic policies, one action dimension at a time. We also introduce two novel methods to improve exploration in DRL-based combinatorial optimization, building on ideas of entropy regularization and distribution initialization. Despite their dramatically reduced complexity, we demonstrate that discovered symbolic policies outperform seven state-of-the-art DRL algorithms in terms of average rank and average normalized episodic reward across eight benchmark environments.

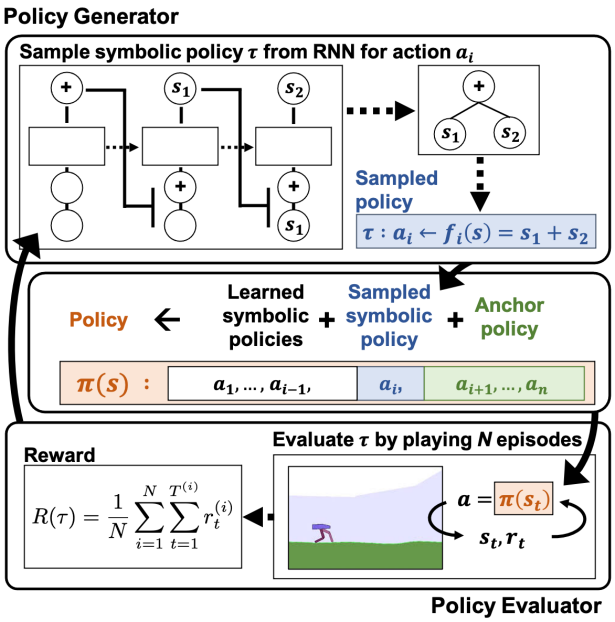

Figure 1: Algorithm overview. (Top) The *Policy Generator* samples an example expression. (Middle) The current policy is a construct of previously learned symbolic policies, the current sample, and an anchor model. (Bottom) The *Policy Evaluator* applies the policy to the environment, and the reward is used to train the *Policy Generator*. The algorithm learns actions sequentially until all n actions are symbolic and the anchor model is discarded.

1. Introduction

Deep reinforcement learning (DRL) has shown remarkable success in the last few years in solving a wide-range of difficult control problems (Collins et al., 2005; Mnih et al., 2015; 2016; Pettit et al., 2019; Glatt et al., 2020). DRL owes much of its success to recent advances in training deep neural networks (NNs), which are commonly employed as function approximators for an RL policy (Rumelhart et al., 1985; Hinton et al., 2012; LeCun et al., 2015). However, NN-based policies exhibit complex functional forms, involving thousands of nested non-linear operators and affine transformations. This complexity poses a significant barrier to deployment of DRL policies in real-world settings due to the difficulty to understand, verify, trust, and predict the behavior of the RL agent. These challenges are particularly relevant for medical domains, in which "black-box"

[*]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.

NN-based models are unacceptable (Tu, 1996; Dayhoff & DeLeo, 2001; London, 2019).

In contrast, traditional approaches in mathematical physics and control theory often yield simple controllers with compact functional forms. Such controllers can be remarkably effective, perhaps because they leverage the "unreasonable effectiveness of mathematics;" that is, their use of symbolic operators captures the regularities and general principles found in nature (Wigner, 1995). From the learning perspective, the use of symbolic operators can be regarded as enforcing a strong form of regularization, one that constrains the model to be expressed in a succinct symbolic form.

In light of this, we propose a learning framework called Deep Symbolic Policy (DSP) in which we directly search the space of tractable mathematical expressions to be used as control policies for RL environments with continuous action spaces. Learning policies represented by tractable mathematical expressions affords several desirable features: (1) **Interpretability.** Insights from concise mathematical expressions can often be gleaned by inspection. Further, we demonstrate that when transition dynamics are known, formal stability analysis can be conducted. (2) **Generalizability.** As we will demonstrate, symbolic policies can be effective at generalizing to continuous-time settings, even when trained in discretized, simulated environments. (3) **Deployability.** Symbolic policies are "small and fast"; that is, they are easily deployable in real-world settings where the need for specialized hardware, the memory footprint of the policy, and latency in executing the policy can be crucial limiting factors. (4) **Transparency** and **Verifiability.** In contrast to neural network-based policies, which can be difficult to reproduce, symbolic policy performance is easily verifiable using just a few lines of code, and the policy exhibits readily predictable behavior. (5) **Performance.** One might expect a notable reduction in performance when using symbolic policies compared to highly complex NN-based policies. However, we demonstrate that symbolic policies exhibit surprisingly robust performance, competitive with NN-based policies.

As depicted in Figure 1, our approach involves a Policy Generator that creates mathematical expressions to be used directly as control policies, and a Policy Evaluator that evaluates the policies by running episodes in the simulated environment. The reward from the environment (inner RL loop) is then used as a learning signal to train the Policy Generator via policy gradients (outer RL loop). To scale the approach to problems with multiple actions, we develop a novel model distillation technique that leverages pre-trained DRL policies to "anchor" the symbolic policy being learned. The NN-based anchor model serves as a temporary surrogate that is removed one dimension at a time; the final product is a fully symbolic policy without any NN-based dimension.

We demonstrate our approach on a series of continuous control benchmark environments. Despite being dramatically less complex than NN-based DRL policies, the discovered symbolic policies achieve the highest normalized episodic reward, highest average rank, and lowest worst-case rank across eight benchmark environments compared to seven DRL algorithms. We summarize our contributions as:

- A gradient-based approach that searches the space of symbolic control policies directly within the RL loop.
- A novel model distillation algorithm to scale to environments with multidimensional action spaces.
- Two novel exploration techniques applicable to using DRL for combinatorial optimization: a *hierarchical entropy regularizer* to avoid premature convergence on the first few tokens, and a *soft length prior* to promote exploration over sequence lengths.
- Discovering symbolic policies that outperform standard DRL algorithms in terms of average normalized episodic reward across benchmark environments.
- Provably stable symbolic policies (with respect to variations in initial conditions) for the three environments in which transition dynamics are known.

2. Related Work

Interpretability in RL. Recently, a number of works have been proposed to address interpretability in RL via symbolic representation of policies. Several studies learn policies represented by decision trees (Ernst et al., 2005; Gupta et al., 2015; Liu et al., 2018; Bastani et al., 2018; Coppens et al., 2019; Roth et al., 2019). A common approach for this is model compression, or distilling a DRL model into a decision tree policy (Gupta et al., 2015; Liu et al., 2018; Bastani et al., 2018; Coppens et al., 2019). Decision tree-based approaches apply to control environments with discrete action spaces.

Several works have been proposed to use *symbolic regression* as a means of obtaining symbolic policies (Kubalík et al., 2017; Hein et al., 2018). Kubalík et al. (2017) use genetic programming to symbolically approximate a policy obtained via classical control methods. However, this approach requires access to the underlying dynamics equations. This limitation is removed in Hein et al. (2018), where the authors proposed to learn a world model and use it to guide a symbolic regression algorithm based on genetic programming. In their results, the discovered symbolic polices suffered from a decrease in performance when compared to a NN policy trained using a single RL algorithm. Recently, Verma et al. (2018) use program synthesis to learn explainable policies, relying on model distillation from a

pre-trained DRL policy. In (Wilson et al., 2018), the authors evolve effective programs for Atari game playing using Cartesian Genetic Programming. While their strategies are fully interpretable, they might end up in local optima and avoid more complex strategies.

The model distillation and regression-based approaches discussed above share a common issue: there is an *objective mismatch* between the training objective and the evaluation metric. Namely, training uses some form of supervised learning—using data sampled from a pre-trained model—to "mimic" a pre-trained "oracle" policy. However, the final evaluation metric is the RL objective: obtaining high reward when executed in the environment. We demonstrate that regression-based approaches often lead to catastrophic failure. In contrast, DSP is trained directly by the reward coming from the environment; thus, there is no objective mismatch. While DSP includes an "anchor" model for environments with multidimensional action spaces, the learning signal is still *always* directly tied to to the symbolic policy's performance in the environment.

AutoML. DSP also has parallels to automated machine learning (AutoML) and structure optimization methods, in which an autoregressive RNN is used to define a distribution over discrete objects, followed by RL to optimize this distribution under a black-box performance metric (Zoph & Le, 2016; Bello et al., 2017; Ramachandran et al., 2017; Abolafia et al., 2018; Pham et al., 2018). More recently, Petersen et al. (2021) proposed an RNN-based approach for symbolic regression and a risk-seeking policy gradient that optimizes for best-case performance. Here, we build upon these combinatorial optimization frameworks by introducing two novel methods to improve exploration. Further, our anchoring algorithm allows us to jointly optimize multiple dependent discrete objects (i.e. multiple action dimensions), whereas works described above are limited to single-object tasks.

3. Deep Symbolic Policy

As illustrated in Figure 1, our overall approach involves two main components: 1) the **Policy Generator**, which creates mathematical expressions (represented as a sequence of tokens) to be used as control policies in an RL environment, and 2) the **Policy Evaluator**, which examines the performance of the generated policy by playing multiple episodes of the environment and outputting the average episodic reward. The output from the Policy Evaluator is then used as a reward signal to train the Policy Generator using a risk-seeking policy gradient (Petersen et al., 2021). Repeating this process, the Policy Generator is iteratively trained to generate better-performing policies.

Pseudocode for DSP with the anchoring algorithm is provided in the Appendix. Source code is made available at `https://www.github.com/brendenpetersen/deep-symbolic-optimization`.

Policy Generator. The Policy Generator models a distribution over mathematical expressions. Expressions can be sampled and directly used as control policies for an RL environment. Each sampled expression is a function $f : \mathcal{S} \to \mathbb{R}$, where $\mathcal{S}$ is the environment's observation space. Before we describe the generative process for f, we first note that expressions can be represented as *symbolic expression trees*, a type of binary tree in which internal nodes are mathematical operators and terminal nodes are input variables or constants. Operators may be unary (i.e. one argument, such as sine) or binary (i.e. two arguments, such as multiply). Further, we can represent an expression tree as a sequence of nodes by using its pre-order traversal (i.e. by visiting each node depth-first, then left-to-right). This allows us to generate an expression tree sequentially while still maintaining a one-to-one correspondence between the tree and its traversal. Thus, we define our search space as a discrete sequence of "tokens" representing operators, input variables, and constants. Tokens are selected from a library $\mathcal{L}$, e.g. $\{+, \times, \sin, s_1, s_2, 0.1, 5.0\}$, where s_i corresponds to the ith dimension of an environment's observation space. We denote the sequence of tokens as τ, whereas its instantiation as an expression is denoted f.

Now that the search is reduced to a discrete sequence, we leverage an idea from AutoML works that optimize distributions over sequences using an autoregressive recurrent neural network (RNN) (Zoph & Le, 2016; Ramachandran et al., 2017; Bello et al., 2017; Pham et al., 2018; Petersen et al., 2021). The sampling process begins with an empty tree. The RNN emits a categorical distribution over the tokens in $\mathcal{L}$, a token is sampled, and it is added to the expression tree. Typically, the next input to the RNN would be a representation of the previously sampled token. To better capture the hierarchical nature of the expression tree, we instead feed a representation of the *parent* and *sibling* token as the next input to the RNN (an "empty" token is used for nodes without a parent or sibling). This process repeats until the expression tree is complete (i.e. all tree branches reach terminal nodes).

Thus, each token is sampled from a categorical distribution (i.e. the RNN emission) that is conditionally independent given the previous tokens (i.e. RNN inputs and cell state). The likelihood for the ith token of the traversal τ_i is given by: $p(\tau_i|\tau_{1:(i-1)}; \theta) = \psi^{(i)}_{\mathcal{L}(\tau_i)}$, where θ are the RNN parameters, $\psi^{(i)}$ is the ith emission of the RNN (corresponding to the probabilities over tokens), and $\mathcal{L}(\tau_i)$ is the index in $\mathcal{L}$ corresponding to token τ_i. The likelihood of the entire sampled expression τ (used in the policy gradient during training) is computed using the chain rule of conditional probabil-

ity: $p(\tau|\theta) = \prod_{i=1}^{|\tau|} p(\tau_i|\tau_{1:(i-1)};\theta) = \prod_{i=1}^{|\tau|} \psi^{(i)}_{\mathcal{L}(\tau_i)}$. An example of the sampling process is shown in Figure 1(Top).

In situ constraints. A benefit to autoregressive sampling is the ability to efficiently encode domain knowledge by imposing constraints directly on the search space. Specifically, we can introduce a broad class of in situ constraints in which one only has to be able to determine which tokens are disallowed at any point along the traversal. This is done by simply zeroing out the probability of selecting a token that would violate a constraint before sampling. This process ensures that all samples adhere to all constraints in situ (concurrently with sampling), without the need to reject samples post hoc. To improve the interpretability of the obtained expressions, we impose several constraints: (1) The length of each expression is constrained to fall between a pre-defined minimum and maximum length. (2) We prohibit the child of an unary operator to be the inverse of that operator, such as $\log(\exp(\cdot))$. (3) For simplicity, we prohibit nested trigonometric operators, such as $\sin(1 + \cos(\cdot))$. (4) To prevent trivial policies, we ensure each expression includes at least one input variable.

Policy Evaluator. Given a sequence τ from the Policy Generator, we instantiate the corresponding mathematical expression f. We then employ f directly as the control policy for a reinforcement learning environment: $a = f(s)$ (we address multi-action environments later). To evaluate the quality of the policy, we run multiple episodes of the environment (inner RL loop) and compute the average episodic reward inside the environment. This result is then used as the reward signal to train the Policy Generator using RL (outer RL loop). The Policy Generator's reward function is given by:

$$R(\tau) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T^{(i)}} r_t^{(i)},$$

where $T^{(i)}$ is the length of episode i, N is the number of episodes, and $r_t^{(i)}$ is the instantaneous reward of episode i at time step t when using the symbolic policy corresponding to τ.

Training with risk-seeking policy gradients. The reward function from the Policy Evaluator is non-differentiable, as it comes from the control environment. Thus, we turn to RL to optimize the Policy Generator. Typically, autoregressive models with black-box reward functions use the standard REINFORCE policy gradient (Williams, 1992; Zoph & Le, 2016; Ramachandran et al., 2017; Bello et al., 2017), which optimizes the objective given by:

$$J(\theta) = \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[R(\tau)\right]. \tag{1}$$

Notably, optimizing $J(\theta)$ maximizes *expected* reward under the distribution. However, as in most AutoML tasks, we seek the *single or few best* performing samples from the distribution. That is, in the end, we seek the single best-performing symbolic policy sampled from our Policy Generator. Thus, we adopt the risk-seeking policy gradient by Petersen et al. (2021), a modified policy gradient that optimizes for best-case performance. Specifically, the objective function is given by:

$$J_{\text{risk}}(\theta;\epsilon) = \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[R(\tau) \mid R(\tau) \geq R_\epsilon(\theta)\right], \tag{2}$$

where ϵ is a hyperparameter controlling the degree of risk-seeking and $R_\epsilon(\theta)$ is the $(1-\epsilon)$ quantile of rewards. Similar to the standard REINFORCE policy gradient, the gradient of this objective can be approximated using a simple Monte Carlo estimate (see Petersen et al. (2021) for details).

Constant optimization. The library $\mathcal{L}$ includes pre-specified real-valued constants as tokens (specifically, 0.1, 1.0, and 5.0). The Policy Generator can compose these tokens using various operators to generate new values; however, it is unlikely to find locally optimal constant values. To further optimize the learned symbolic policies, we fine-tune the real-valued constants (see the Appendix for details). We perform constant optimization only once per environment on the best-performing symbolic policy.

4. Scaling Up to Multiple Action Dimensions

So far, we have discussed DSP in the context of RL environments with a single-dimensional action space, i.e. $\mathcal{A} \in \mathbb{R}$. However, many environments exhibit n-dimensional action spaces, i.e. $\mathcal{A} \in \mathbb{R}^n$. One possible solution to scale to multi-dimensional action spaces would be to use the Policy Generator to generate n expressions sequentially. A single sample would then correspond to $\{f_1, \ldots, f_n\}$, where the ith action a_i is given by $a_i = f_i(s)$. However, this approach scales very poorly with n. Consider a library $\mathcal{L}$ of size $|\mathcal{L}|$, a maximum sequence length (per expression) of k, and an environment with $\mathcal{A} \in \mathbb{R}^n$. The size of this combinatorial action space is upper-bounded by $\mathcal{O}(|\mathcal{L}|^{nk})$, i.e. it scales exponentially with n for a problem that already scales exponentially with k.

To circumvent this combinatorial explosion, we propose a novel model distillation approach that reduces the search to n sub-problems each with size $\mathcal{O}(|\mathcal{L}|^k)$. We leverage an existing pre-trained policy model (e.g. a neural-network based policy trained using an existing DRL algorithm) which we refer to as an *anchor model* $\Psi : \mathcal{S} \to \mathcal{A}$. The proposed process for learning a purely symbolic policy in an environment with $\mathcal{A} \in \mathbb{R}^n$ then involves n sequential—but dependent—rounds of DSP. In the first round, DSP learns a "sub-policy" $f_1 : \mathcal{S} \to \mathbb{R}$, which is used to determine the value of the first action: $a_1 = f_1(s)$. In the Policy Evaluator, actions for the remaining actions are determined using the anchor model: $a_i = \Psi(s)_i \forall\, i \in \{2, \ldots, n\}$, where $\Psi(s)_i$ repre-

sents the ith element of the vector $\Psi(s)$. The final result of this first round of DSP training is $f_1^\star$, the best expression found during training. For the next round of DSP, we treat $f_1^\star$ as *fixed*, which we emphasize using the notation $\bar{f}_1$.

In the second round of DSP, the sub-policy f_2 (used to compute a_2) becomes the learning target; $\bar{f}_1$ is used to compute a_1, and the anchor model is used to compute the remaining actions $a_3, \ldots, a_n$. This process continues until the policy is fully symbolic. In general, for round i of DSP, the first $i-1$ actions are computed using fixed symbolic policies $\bar{f}_1, \ldots, \bar{f}_{i-1}$, the learning target f_i is used to compute a_i, and the anchor model is used to compute the remaining actions $a_{i+1}, \ldots, a_n$. Thus, in the final round of DSP (just as in the only round of DSP for single-dimensional action spaces), the anchor model is not used at all. In essence, the anchor model acts like a "ladder": each "rung" (i.e. action dimension) is discarded after the algorithm has climbed it; by the end of the algorithm, the entire ladder is removed.

Pseudocode for DSP including the anchoring algorithm is provided in the Appendix. The intuition behind the anchoring process is that it allows DSP to focus learning on a single action dimension at a time, while still "grounding" or "anchoring" the learning process for all other action dimensions using Ψ and/or $\bar{f}_i$.

5. Improving Exploration

The RNN-based policy gradient setup described above is a powerful way to search in the combinatorial space of expression trees (Bello et al., 2017). However, we show that it can suffer from an *early commitment* phenomenon and from *initialization bias*, both of which limit exploration. To address these issues, we present two novel exploration techniques for RNN-based neural-guided search: *hierarchical entropy regularizer* and *soft length prior*.

We demonstrate the efficacy of each of these two contributions by performing ablation experiments (see Appendix).

Hierarchical entropy regularizer. Policy gradient methods typically include a "bonus" to the loss function proportional to the entropy at each step of autoregressive sampling (Haarnoja et al., 2018; Abolafia et al., 2018). More precisely, the term

$$\mathcal{H}(\theta) = \eta \, \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_{i=1}^{|\tau|} H[p(\tau_i | \tau_{1:(i-1)}; \theta)] \right] \quad (3)$$

is added to (1) or (2), where $H[p(X)] = -\sum_{x \in X} p(x) \log p(x)$ is the entropy and η is a hyperparameter controlling the importance of the entropy term. The entropy regularizer (3) promotes exploration and helps prevent the RNN from converging prematurely to a local optimum. Notably, (3) is simply a sum across time steps i; thus, all time steps contribute equally.

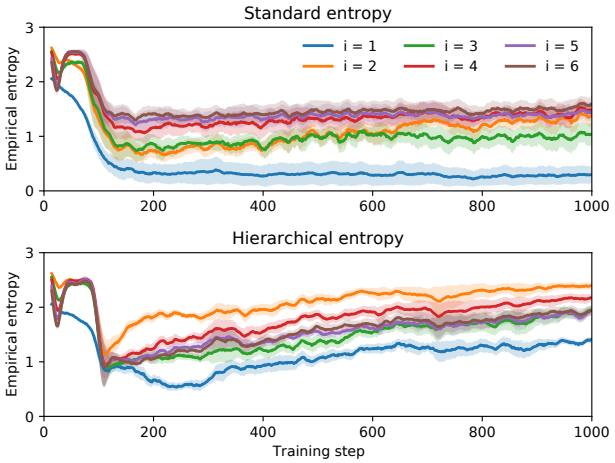

Figure 2: Evolution of the empirical entropy of the set $\{\tau_i^{(j)}\}_{j=1}^M$ for $1 \leq i \leq 6$ with standard $\mathcal{H}$ (top) and hierarchical $\mathcal{H}_\gamma$ (bottom) entropy regularizer on InvertedPendulumSwingup with $\eta = 0.02$. Results are averaged over 10 independent runs.

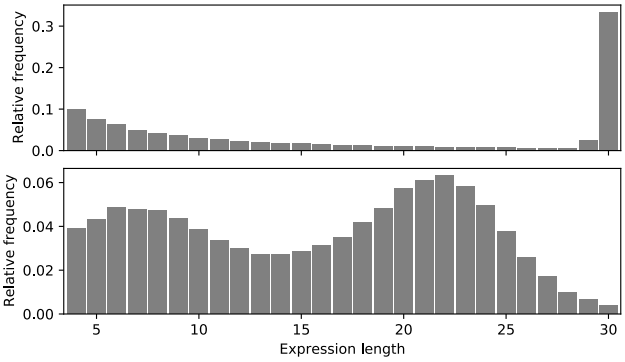

Figure 3: Initial distributions over expression lengths under the Policy Generator without (top) or with (bottom) the soft length prior ($\lambda = 10, \sigma^2 = 20$).

When optimizing discrete sequences (or navigating RL environments with deterministic transition dynamics), each sequence can be viewed as a path through an underlying search tree. Due to the hierarchical nature of this tree, it is much easier to achieve high entropy at later time steps, which are far less explored. This causes the sum in (3) to be concentrated in the later terms, while the earliest terms can quickly approach zero entropy. When this happens, the RNN stops exploring early tokens entirely and thus entire branches of the search space, which can greatly hinder performance. This phenomenon, which we refer to as the "early commitment problem," can be observed empirically, as shown in Figure 2 (top). Note that the entropy of the first token of the sequence drops to zero early on in training.

Table 1: Best discovered symbolic policies before (DSP) and after (DSP^{o}) optimizing constants.

Environment	DSP	DSP^{o}
CartPole	$a_1 = 10s_3 + s_4$	$a_1 = 10.03s_3 + 0.45s_4$
MountainCar	$a_1 = -\frac{0.62}{\log(s_2)}$	$a_1 = -\frac{0.601}{\log(s_2)}$
Pendulum	$a_1 = -2s_2 - \frac{8s_2+2s_3}{s_1}$	$a_1 = -7.08s_2 - \frac{13.39s_2+3.12s_3}{s_1} + 0.27$
InvDoublePend	$a_1 = -12.2s_8$	$a_1 = -12.23s_8$
InvPendSwingup	$a_1 = s_1 + 5s_4 + s_5 + s_6 + \sin(s_2) + \sin(s_2 + s_4 + s_5) + 0.19$	$a_1 = s_1 + 5s_4 + s_5 + s_6 + \sin(s_2) + \sin(s_2 + s_4 + s_5) + 0.21$
LunarLander	$a_1 = -10s_2 + \sin(s_3) - 14s_4 - 1.99$ $a_2 = -5.79\frac{s_4}{s_6 - s_3}$	$a_1 = -5.99s_2 + 0.76\sin(s_3) - 9.80s_4 - 1.35$ $a_2 = -3.49\frac{s_4}{s_6 - s_3}$
Hopper	$a_1 = \exp\left(\frac{s_6}{s_{10}}\sec\left(\frac{s_1 s_2 s_{14}}{s_1 + s_4 - s_8}\right)\right)$ $a_2 = -5s_4 - 2s_6 - 6s_8 - s_{11} + \cos(s_4)$ $a_3 = \frac{\cos(s_{13})}{s_{11} + \sin(s_1)}$	$a_1 = 1.09\exp\left(\frac{s_6}{s_{10}}\sec\left(\frac{s_1 s_2 s_{14}}{s_1 + s_4 - s_8}\right)\right) - 0.02$ $a_2 = -6.22s_4 - 2.49s_6 - 7.47s_8 - 1.24s_{11} + 1.24\cos(s_4) - 0.03$ $a_3 = 1.03\frac{\cos(s_{13})}{s_{11} + \sin(s_1)}$
BipedalWalker	$a_1 = s_1 - s_8 - s_9 - 2s_{11} + s_{24}$ $a_2 = \frac{\sin(2s_3 - s_7)}{s_{22} - \sin(s_3) + \cos(s_7)} - s_7$ $a_3 = s_3 - s_{10} - \sin(s_{12}) - \cos(2s_{10})$ $a_4 = \frac{s_{10}}{s_2}\left(s_2^2 - \frac{e}{s_2} + \log(s_2) + 5\right)$	$a_1 = 0.03s_1 - 0.03s_8 - 0.03s_9 - 0.1s_{11} + 0.03s_{24} + 0.11$ $a_2 = \frac{0.9\sin(3.12s_3 - s_7)}{s_{22} - \sin(s_3) + \cos(s_7)} - 0.9s_7 - 0.4$ $a_3 = 1.14s_3 - 1.14s_{10} - 1.14\sin(s_{12}) - 1.14\cos(1.95s_{10}) - 0.22$ $a_4 = 0.18\frac{s_{10}}{s_2}\left(s_2^2 - \frac{e}{s_2} + \log(s_2) + 3.28\right) + 3.24$

We propose a simple change to combat the early commitment problem: replacing the sum in (3) with a weighted sum whose weights decay exponentially:

$$\mathcal{H}_\gamma(\theta) = \eta \mathbb{E}_{\tau \sim p(\tau|\theta)}\left[\sum_{i=1}^{|\tau|} \gamma^{i-1} H[p(\tau_i | \tau_{1:(i-1)}; \theta)\right]. \quad (4)$$

Using (4) encourages the RNN to perpetually explore even the earliest terms. In Figure 2 (bottom), we demonstrate the effectiveness of this hierarchical entropy regularizer in ensuring that the RNN maintains diversity across tokens during the course of training.

Soft length prior. Before training begins, using zero-weight initializers, the RNN emissions $\psi^{(i)}$ start at zero. Thus, the probability distribution over tokens is uniform: $p(\tau_i) = \frac{1}{|\mathcal{L}|} \forall\, \tau_i \in \mathcal{L}$. We can inform this starting distribution by including a prior, i.e. by adding a logit vector $\psi_\circ$ to each RNN emission $\psi^{(i)}$. For example, we can ensure that the prior probability of choosing a binary, unary, or terminal token is equal (regardless of the number of each type of token in the library) by solving for $\psi_\circ$ below:

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

where the tokens corresponding to $\psi_\circ$ are sorted by decreasing arity, $(\cdot)_n$ denotes that element $(\cdot)$ is repeated n times, $\|$ denotes vector concatenation, and n_2, n_1, and n_0 are the number of binary, unary, and terminal tokens in $\mathcal{L}$, respectively. The solution is:

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

where c is an arbitrary constant (see Appendix for proof).

Under the prior $\psi_\circ$, the distribution over expression lengths is heavily skewed toward longer expressions. In particular, the expected expression length is $\mathbb{E}_{\tau \sim \psi_\circ}[|\tau|] = \infty$ (see Appendix for proof). In practice, a *length constraint* is applied; however, empirically this results in the vast majority of expressions having length equal to the maximum allowable length. We show this empirically in Figure 3 (top). This strong bias makes it very difficult for the distribution to *learn* an appropriate expression length. To provide this capability, we introduce an additional *soft length prior* to the RNN's ith emission:

$$\psi_\circ = \left(\frac{-(i-\lambda)^2}{2\sigma^2}\mathbb{1}_{i>\lambda}\right)_{n_2} \| (0)_{n_1} \| \left(\frac{-(i-\lambda)^2}{2\sigma^2}\mathbb{1}_{i<\lambda}\right)_{n_0},$$

where the tokens corresponding to $\psi_\circ$ are sorted by decreasing arity, and λ and σ are hyperparameters. In probability

Table 2: Performance comparison of symbolic policies discovered by DSP, DSP^o, regression, and Zoo policies for seven different DRL algorithms. Values are episodic rewards averaged over 1,000 episodes using a held-out set of environment seeds (identical across algorithms). $\overline{\text{Zoo}}$ represents the average over the seven Zoo policies. †As an additional Regression baseline for MountainCar, we evaluated the best symbolic policy reported by Hein et al. (2018), which has an average episodic reward of 90.40.

Environment	DSP	DSP^o	Regression	DDPG	TRPO	A2C	PPO	ACKTR	SAC	TD3	$\overline{\text{Zoo}}$
CartPole	999.59	1000	211.82	1000	1000	1000	993.94	1000	971.78	997.98	994.81
MountainCar	99.09	99.11	95.16†	91.77	93.95	93.63	92.56	93.77	90.40	93.93	92.86
Pendulum	-160.5	-155.4	-1206.9	-169.0	-147.6	-162.2	-154.8	-211.2	-159.3	-147.1	-164.4
InvDoublePend	9148.2	9149.9	637.2	8855.1	8854.8	8951.9	9225.5	7554.1	9313.8	9357.8	8873.3
InvPendSwingup	891.84	891.90	-19.21	891.34	892.51	67.52	853.38	890.34	891.47	889.33	767.98
LunarLander	251.66	261.36	56.08	246.24	168.79	227.08	225.12	245.39	272.65	225.35	230.09
Hopper	2090.2	2122.4	47.35	1632.7	2583.4	1925.1	2439.7	2456.7	2455.0	2741.9	2319.2
BipedalWalker	264.39	311.78	-110.77	94.21	311.08	241.01	286.20	299.32	307.26	310.19	264.18
Average rank	**2.63**		8.13	5.63	3.25	5.63	5.63	4.88	4.50	3.50	
Worst rank	**6**		9	8	8	8	7	8	9	**6**	
Average $\bar{R}_{\text{ep}}$	**0.96**		0.07	0.75	0.85	0.72	0.85	0.86	0.85	0.90	

space, $\psi_\circ$ is a multiplicative Gaussian function applied to either binary tokens ($i > \lambda$) or terminal tokens ($i < \lambda$). Thus, it discourages expressions from being either too short or too long. Figure 3 (bottom) shows that including this prior results in a much smoother a priori distribution over expression lengths. In contrast to the length constraint (i.e. *hard* length prior) that *forces* each expression to fall between a pre-specified minimum and maximum length, the soft length prior affords the Policy Generator the ability to *learn* the optimal length; as shown in the Appendix, this greatly improves learning.

6. Results and Discussion

Experimental setting. The Policy Generator is an RNN comprising a single-layer LSTM with 32 hidden units. For each action in the control task, we perform 3 independent training runs of DSP with different random seeds, selecting the best symbolic policy at the end of training. Thus, for environments with $\mathcal{A} \in \mathbb{R}^n$, we perform $3n$ training runs and select the single best policy. Since constant optimization is computationally expensive, we perform it only once per environment on the best found symbolic policy. All tasks use the library $\mathcal{L} = \{+, -, \times, \div, \sin, \cos, \exp, \log, 0.1, 1.0, 5.0, s_1, \ldots, s_n\}$, where s_i is the ith dimension of the environment's observation. When computing actions, the Policy Evaluator replaces any infinite or undefined actions with zero, e.g. the symbolic policy $a_1 = \log(s_1)$ returns $\log(s_1)$ for $s_1 > 0$ and 0 otherwise. We constrain the length of each expression to fall between 4 and 30 tokens, inclusive.

Evaluation for all algorithms (DSP and all baselines) is based on the average episodic reward across 1,000 episodes with different random environment seeds. Notably, we evaluate all policies on the same held-out set of 1,000 random environment seeds. The environment seed is largely used to determine the environment's initial state, and some seeds can be significantly more challenging than others. Thus, this step is crucial to ensure fair comparison across algorithms.

Performance comparisons. We evaluate DSP on eight benchmark control tasks: five single-action environments (CartPole, MountainCar, Pendulum, InvertedDoublePendulum, and InvertedPendulumSwingup) and three multi-action environments (LunarLander, Hopper, and BipedalWalker)[1]. The best symbolic policies found for each environment are reported in Table 1. We include results both before and after constant optimization (labelled DSP and DSP^o, respectively).

The "Regression" baseline is computed using the following workflow: (1) we generate an offline dataset of observation-action trajectories from the top-performing pre-trained Zoo policy (described below); (2) we perform *symbolic regression* on the offline dataset using deep symbolic regression (Petersen et al., 2021) and select the expression with the lowest error for each action; (3) we evaluate the best symbolic expressions in the environments described above. Additional details for this baseline are provided in the Appendix.

Aside from MountainCar, results from the Regression baseline are catastrophic. This demonstrates the subtle yet cru-

[1]Specifically, we use CartPoleContinuous-v0 from https://gist.github.com/iandanforth; MountainCarContinuous-v0, Pendulum-v0, LunarLanderContinuous-v2, and BipedalWalker-v2 from OpenAI Gym (Brockman et al., 2016); and InvertedDoublePendulumBulletEnv-v0, InvertedPendulumSwingupBulletEnv-v0, and HopperBulletEnv-v0 from PyBullet (Coumans & Bai, 2016).

cial flaw with a regression-based approach: there is an objective function mismatch between the regression objective (minimizing prediction error) and the control objective (obtaining high reward in the environment). In other words, *small errors in regression do not necessarily correspond to small decreases in performance when evaluated in the environment*. While it can be effective for simple environments like MountainCar (as verified by Hein et al. (2018)), we believe that a regression-based approach is ultimately fundamentally flawed.

As a much stronger baseline, we compare to seven state-of-the-art DRL algorithms: DDPG, TRPO, A2C, PPO, ACKTR, SAC, and TD3 (Lillicrap et al., 2015; Schulman et al., 2015; Mnih et al., 2016; Schulman et al., 2017; Wu et al., 2017; Haarnoja et al., 2018; Fujimoto et al., 2018). We leverage Zoo, an open-source repository containing pre-trained DRL policies that have been individually tuned for each environment.[2]

In Table 2, we report the evaluation average episodic rewards for DSP, DSP^{o} (i.e., DSP with optimization of constants), the Regression baseline, and the seven DRL baselines. To enable comparison across environments, we report average rank, worst-case rank, and average normalized episodic reward, computed as $\bar{R}_{\text{ep}} = (\text{score} - \min)/(\max - \min)$, where $\min$ and $\max$ are computed for each environment across all algorithms. Additional results for the intermediate hybrid policies in the multi-action environments are provided in the Appendix. Remarkably, despite the dramatically lower complexity of symbolic policies, DSP performs well across environments, yielding the highest average rank, highest average normalized episodic reward, and best worst-case rank (tied with TD3) compared to all baselines. Further, DSP outperforms the average across Zoo policies ($\overline{\text{Zoo}}$) for all environments except Hopper.

The reduced complexity of the symbolic policies discovered by DSP can be seen very clearly in Figure 4 for three environments. Analogous heatmaps for all environments are provided in the Appendix. Videos of the discovered symbolic policies in Table 1 deployed in each environment are available in the multimedia Appendix.

Stability analysis. In this section, we analyze the stability of the discovered symbolic policy for CartPole. We repeat this analysis for Pendulum and MountainCar, the other two environments with state transition dynamics that can be written as tractable dynamical systems, in the Appendix. For each environment, we apply the discovered policy into the corresponding dynamical system and evaluate the stability of the system at equilibrium points, using standard eigenvalue analysis (Lee & Markus, 1967; Brunton & Kutz, 2019). For comparison, we also present the analysis of the uncontrolled systems (i.e., taking the action to be identically zero over time). We demonstrate that the CartPole policy is stable in the *continuous* system, even though it was only trained in the discretized version of the transition dynamics (i.e. the discrete time Markov Decision Process emerging from first-order Euler time discretization of the differential equations). This suggests that the use of symbolic policies trained in discrete-time settings (i.e. simulated environments) can still generalize well even in their real-world continuous time counterparts (for which the state transition dynamics are deterministic and described by ordinary differential equations),

The continuous dynamics of the CartPole environment are

[2] A subset of environment-algorithm combinations did not have pre-trained Zoo policies; in these cases, we manually trained Zoo policies using training scripts from the Zoo repository.

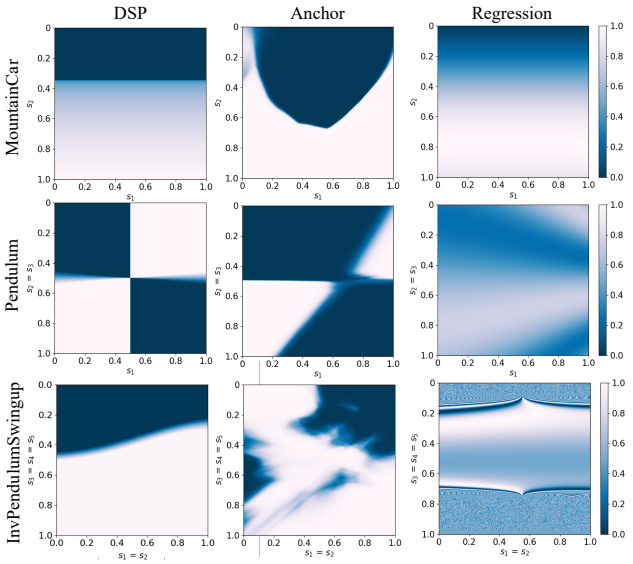

Figure 4: Heatmaps of actions computed from the anchor model (middle column), symbolic policy learned by DSP (left column), and the symbolic policy learned by regression on data generated from the anchor model (right column) in select environments.

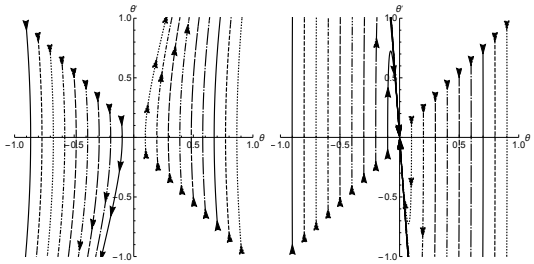

Figure 5: Phase portrait of the uncontrolled (left) and controlled (right) cart pole system ($x = 0, x' = 0$). The controlled system is using the policy discovered by DSP.

defined in (Barto et al., 1983):

$$x''(t) = \frac{8u(t) + 2m\sin(\theta(t))\left(4l\theta'(t)^2 - 3g\cos(\theta(t))\right)}{8m_c - 3m\cos(2\theta(t)) + 5m},$$
$$\theta''(t) = \frac{g\sin(\theta(t)) - \frac{\cos(\theta(t))\left(a_1(t) + lm\theta'(t)^2\sin(\theta(t))\right)}{m_c+m}}{l\left(\frac{4}{3} - \frac{m\cos^2(\theta(t))}{m_c+m}\right)}, \quad (5)$$

where g is gravitational acceleration, m_c is the mass of cart, m is the mass of pole, l is the half-pole length, $x(t)$ is the position of the cart, $\theta(t)$ is the angle with respect the vertical axis and $a_1(t)$ is the action. The task in Cart-Pole is to stabilize system (5) around the equilibrium point $s_{\text{eq}} = (x, x', \theta, \theta') = (0, 0, 0, 0)$. The point s_{eq} is a non-stable equilibrium of the system, as one can check by taking $a_1(t) = 0$ in (5), linearizing the system around s_{eq}, and computing the eigenvalues of the resulting matrix. This procedure yields eigenvalues $(-3.97, 3.97, 0, 0)$. Since the second eigenvalue is positive, we conclude, using the Hartman–Grobman theorem (Grobman, 1959; Hartman, 1960), that s_{eq} is an unstable equilibrium. If we instead use the policy discovered by DSP in Table 1 and repeat the above procedure, we obtain eigenvalues $(-29.61, -14.3, 0, 0)$. Since all eigenvalues are non-positive, we conclude, again using the Hartman–Grobman theorem, that s_{eq} is a stable equilibrium of the controlled system. Therefore, the policy discovered by DSP controls the system by transforming s_{eq} from an unstable equilibrium into a stable equilibrium. We illustrate this in Figure 5 (left), which shows the robustness of the discovered policy over a wide range of initial conditions.

7. Conclusion

We introduce *deep symbolic policy*, a method for learning symbolic control policies using neural-guided search. Our framework includes a novel model distillation method to scale the approach to environments with multi-dimensional action spaces, and two novel exploration techniques applicable to DRL for discrete optimization. We demonstrate the approach on eight continuous control tasks, outperforming highly-tuned, pre-trained NN-based policies in terms of average rank and normalized episodic reward. Our generated policies offer the benefits of being readily interpretable, cheaply deployable, highly transparent, and easily reproducible. Moreover, when transition dynamics are available, our symbolic control policies are shown to be provably stable. Our results suggest that symbolic policies can be a strong alternative to NN-based policies, especially when human interpretability is a desired feature or when deployment constraints (e.g. memory and latency requirements) are important design criteria.

8. Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. LLNL-CONF-822825.

References

Abolafia, D. A., Norouzi, M., Shen, J., Zhao, R., and Le, Q. V. Neural program synthesis with priority queue training. arXiv preprint arXiv:1801.03526, 2018.

Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE transactions on systems, man, and cybernetics, 13(5):834–846, 1983.

Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In Advances in neural information processing systems, 2018.

Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. Neural optimizer search with reinforcement learning. In Proc. of the International Conference on Machine Learning, 2017.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

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

Collins, S., Ruina, A., Tedrake, R., and Wisse, M. Efficient bipedal robots based on passive-dynamic walkers. Science, 307(5712):1082–1085, 2005.

Coppens, Y., Efthymiadis, K., Lenaerts, T., Nowé, A., Miller, T., Weber, R., and Magazzeni, D. Distilling deep reinforcement learning policies in soft decision trees. In Proc. of the IJCAI Workshop on Explainable Artificial Intelligence, 2019.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

Dayhoff, J. E. and DeLeo, J. M. Artificial neural networks: opening the black box. Cancer: Interdisciplinary International Journal of the American Cancer Society, 91 (S8):1615–1635, 2001.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research, 6(4):503–556, 2005.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. arXiv:1802.09477, 2018.

Glatt, R., Da Silva, F. L., da Costa Bianchi, R. A., and Costa, A. H. R. Decaf: Deep case-based policy inference for knowledge transfer in reinforcement learning. Expert Systems with Applications, 156:113420, 2020.

Grobman, D. M. Homeomorphism of systems of differential equations. Doklady Akademii Nauk SSSR, 128(5):880–881, 1959.

Gupta, U. D., Talvitie, E., and Bowling, M. Policy tree: Adaptive representation for policy gradient. In AAAI Conference on Artificial Intelligence, 2015.

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

Hartman, P. A lemma in the theory of structural stability of differential equations. Proc. of the American Mathematical Society, 11(4):610–620, 1960.

Hein, D., Udluft, S., and Runkler, T. A. Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence, 76: 158–169, 2018.

Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 14(8), 2012.

Kubalík, J., Alibekov, E., and Babuška, R. Optimal control via reinforcement learning with symbolic policy approximation. IFAC-PapersOnLine, 50(1):4162–4167, 2017.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. Nature, 521(7553):436–444, 2015.

Lee, E. B. and Markus, L. Foundations of optimal control theory. Technical report, Minnesota Univ Minneapolis Center For Control Sciences, 1967.

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

Liu, G., Schulte, O., Zhu, W., and Li, Q. Toward interpretable deep reinforcement learning with linear model u-trees. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2018.

London, A. J. Artificial intelligence and black-box medical decisions: accuracy versus explainability. Hastings Center Report, 49(1):15–21, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. Nature, 518(7540): 529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proc. of the International conference on machine learning, 2016.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. Proc. of the International Conference on Learning Representations, 2021.

Pettit, J. F., Glatt, R., Donadee, J. R., and Petersen, B. K. Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning. arXiv preprint arXiv:1912.03408, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. arXiv:1802.03268, 2018.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. arXiv:1710.05941, 2017.

Roth, A. M., Topin, N., Jamshidi, P., and Veloso, M. Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. arXiv:1907.01180, 2019.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Technical report, California Univ. San Diego La Jolla Inst. for Cognitive Science, 1985.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In Proc. of the International conference on machine learning, 2015.

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

Tu, J. V. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. Journal of clinical epidemiology, 49 (11):1225–1231, 1996.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. arXiv:1804.02477, 2018.

Wigner, E. P. The unreasonable effectiveness of mathematics in the natural sciences. In Philosophical Reflections and Syntheses, pp. 534–549. Springer, 1995.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992.

Wilson, D. G., Cussat-Blanc, S., Luga, H., and Miller, J. F. Evolving simple programs for playing atari games. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 229–236, 2018.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In Advances in neural information processing systems, pp. 5279–5288, 2017.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. arXiv:1611.01578, 2016.