Neural Circuit Architectural Priors for Embodied Control

Anonymous authors
Paper under double-blind review

Abstract

Artificial neural networks coupled with learning-based methods have enabled robots to tackle increasingly complex tasks, but often at the expense of requiring large amounts of learning experience. In nature, animals are born with highly structured connectivity in their brains and nervous systems that enables them to efficiently learn robust motor skills. Capturing some of this structure in artificial models may bring robots closer to matching animal performance and efficiency. In this paper, we present Neural Circuit Architectural Priors (NCAP), a set of reusable architectural components and design principles for deriving network architectures for embodied control from biological neural circuits. We apply this method to control a simulated agent performing a locomotion task and show that the NCAP architecture achieves comparable asymptotic performance with fully connected MLP architectures while dramatically improving data efficiency and requiring far fewer parameters. We further show through an ablation analysis that principled excitation/inhibition and initialization play significant roles in our NCAP architecture. Overall, our work suggests a way of advancing artificial intelligence and robotics research inspired by systems neuroscience.

1 Introduction

Artificial neural networks (ANNs) coupled with learning-based methods have enabled robots to tackle increasingly complex tasks in domains including legged locomotion, grasping, manipulation, and humanoid control (Pierson & Gashler, 2017; Ibarz et al., 2021; Merel et al., 2019a). Training ANNs for computer vision and natural language processing tasks has greatly benefited from the availability of large datasets (Bommasani et al., 2021), but gathering such large-scale data for robotics is challenging. Experience must be gathered through interaction with the environment, often on the target robot itself, which is difficult for multiple reasons including time, human labor, safety, maintenance, and reproducibility (Kroemer et al., 2020). Without improvements in data efficiency, the amount of data needed to train agents will only increase as models become larger in order to tackle more complex tasks (Ibarz et al., 2021).

In nature, animals are born with highly structured connectivity in their brains and nervous systems that has been shaped over millennia by evolution (Zador, 2019; Merel et al., 2019a). In some cases, this innate circuitry confers abilities with little or no learning; in others, it guides the learning process by providing strong inductive biases (Lake et al., 2017). These innate and learned mechanisms act synergistically, enabling most animals to function well soon after birth, while continuing to improve and acquire skills in a data efficient manner, e.g. a horse learning to walk within a couple hours. Moreover, despite species-specific variations, there is a significant amount of shared architecture and design principles, e.g. hierarchical modularity and partial autonomy (Merel et al., 2019a), even between distantly related species. This suggests that structured neural circuits in animals instantiate robust, flexible, and transferable solutions for high-dimensional embodied control.

Intuitively, capturing some of this structure in artificial models may bring robots closer to matching animal performance and efficiency. After all, importance of architectural priors in ANNs is widely appreciated across machine learning domains. In computer vision, convolutional networks are the norm, combining the inductive biases of spatial locality and weight sharing to achieve both learning and parameter efficiency, as well as parallels with the animal visual cortex (Lindsay, 2021). Many layer types, activation functions, and modules have been proposed (Gu et al., 2017), and the

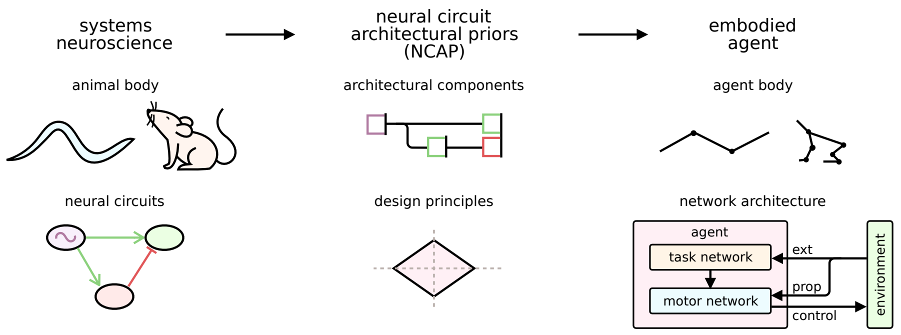


Figure 1: **Method Overview.** NCAP is a set of reusable architectural components and design principles for designing embodied agent network architecures by taking inspiration from biological neural circuits governing motor control.

development of new architectures has yielded increasingly better performance, e.g. LeNet (LeCun et al., 1989), AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2015). In natural language processing, architectural priors have specialized to handle sequences with designs including recurrent networks, e.g. LSTM (Hochreiter & Schmidhuber, 1997), and attention networks, e.g. Transformer (Vaswani et al., 2017). Recently, literature has emerged on neural architecture search (Elsken et al., 2019; Radosavovic et al., 2020) seeking to automate the design of ever better architectures. In robotics, meanwhile, it is still common practice to produce action outputs using fully connected multi-layered perceptron (MLP) architectures (Pierson & Gashler, 2017; Levine et al., 2016; Bin Peng et al., 2020; Heess et al., 2016). What might be some inductive biases useful for motor control?

In this paper, we present **Neural Circuit Architectural Priors** (NCAP), a set of reusable architectural components and design principles for deriving network architectures for embodied control from biological neural circuits. The key idea is to combine the discrete-time formulation of ANNs prevalent in AI research with features of computational neuroscience models like principled excitation/inhibition and intrinsic oscillators. This allows us to design small, sparse networks that mirror neural circuits yet are fully differentiable and therefore compatible with backpropagation-based learning algorithms. In contrast to previous work on neuromechanical models of movement (Sarma et al., 2018; Boyle et al., 2012; Izquierdo & Beer, 2015), our models are more abstract, include learned parameters, and are used to control bodies significantly different from the originals in terms of mechanics, degrees of freedom, and actuators.

We apply NCAP to translate *C. elegans* locomotion circuits into a simulated Swimmer agent, which we train using both reinforcement learning (RL) and evolution strategies (ES). We show that our NCAP architecture achieves comparable asymptotic performance with fully connected MLP architectures, while dramatically improving data efficiency and requiring far fewer parameters. We further show through an ablation analysis that principled excitation/inhibition and initialization play significant roles in our NCAP architecture.

Code and videos are available in the Supplementary Materials.

2 Related Work

A robotic controller ultimately outputs generalized torques $\boldsymbol{\tau}$ to apply at each actuator. A neural network controller can directly output torques (Levine et al., 2016), or it can output task-space positions $\boldsymbol{x}$ or accelerations $\ddot{\boldsymbol{x}}$ that are converted into torques through an analytic controller like operational space control (Khatib, 1987). However, some further level of abstraction is often used to simplify the learning problem (Kroemer et al., 2020).

Trajectory priors encode desired movement through equations of motion. Dynamic Movement Primitives (DMP) (Schaal, 2006; Pastor et al., 2009) use a set of differential equations to implement a stable nonlinear attractor system capable of generating rhythmic and discrete trajectories, which are controlled via low-dimensional parameters specifying the motion's shape and goal. Policies

Modulating Trajectory Generators (PMTG) (Iscen et al., 2019) learn a policy to control a predefined trajectory generator via low-dimensional parameters and also generate a residual term to be added to the trajectory generator's output. For example, to produce legged locomotion, PMTG uses equations of motion composed from a combination of sinusoidal functions and hand-engineered gait patterns, which are parameterized by stride length, walking height, and frequency. Generally, trajectory-centric methods can work well when the equations of motion capture good solutions for the desired movement; however, such equations can be difficult to design and make robust.

Behavioral imitation priors encode desired movement through learnable functions, e.g. ANNs, trained to imitate reference motions, e.g. from motion capture or manual keyframing. Neural Probabilistic Motor Primitives (NPMP) (Merel et al., 2019b) train expert policies to imitate human motion capture trajectories, then compress these experts into a single generalist policy featuring a latent-variable bottleneck, thereby creating a common embedding space that a higher-level controller can use to interpolate and combine various motor primitives. Bin Peng et al. (2020) train expert policies to imitate animal motion capture trajectories using reinforcement learning and further deploy the learned policies on a physical legged robot. Generally, imitation methods bypass the time and manual effort involved in designing equations of motion and have achieved diverse behaviors. This comes at the cost of compiling of reference motions from humans and animals. Further, learning individual expert policies does not make use of the shared structure between movements to learn more efficiently. In animal legged locomotion, for instance, neural circuit studies have suggested that different gait patterns (e.g. walk, trot, bound, gallop) could actually be different emergent dynamic modes of the same pattern generator network (Grillner & El Manira, 2020). In this case, a single policy with the right structure should be able to capture these diverse reference motions.

Architectural priors encode desired movement indirectly through the structure of ANNs, establishing inductive biases to guide learning. Heess et al. (2016) decompose a policy into a low-level "spinal" network with access to only proprioceptive signals (e.g. joint positions) and a high-level "cortical" network with access to exteroceptive signals (e.g. distance to target, vision); this hierarchical architecture was loosely inspired by animal nervous systems. After pre-training the spinal network with a shaped reward, the cortical network was able to learn complex locomotion tasks from sparse rewards by controlling the spinal network, while flat baseline architectures failed. This architecture was later shown to generalize to locomotion with challenging terrains and obstacles, providing greatly improved learning efficiency (Heess et al., 2017). Generally, architectural priors can provide improved efficiency and abstraction while maintaining flexibility. However, the level of flexibility needs to be chosen appropriately. Too much flexibility can lead to learning behaviors that are not naturalistic and infeasible/dangerous to deploy in the real world without some form of regularization (Bin Peng et al., 2020), while too much constraint can impede the learning of diverse movements.

In this work, we develop more constrained architectural priors that should nevertheless in principle be able to scale to diverse movements, since they mirror the mechanisms that animals use. While we use reward-based algorithms to learn parameters (i.e. RL and ES), our architecture is largely orthogonal to the learning algorithm, and we can see it also being useful in a behavioral imitation setting.

3 METHODS

Agent Formalization We consider an agent formalized as a policy function $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ that maps states $\boldsymbol{s}_t$ to actions $\boldsymbol{a}_t$, and which is represented by an ANN parameterized by weights $\boldsymbol{\theta}$. We assume that states and actions are continuous and that the agent is has a body similar to some biological animal.

3.1 ARCHITECTURAL COMPONENTS

Integrator Units Signals in biological neural circuits are processed and integrated (as in "integrate-and-fire"), often by simple neurons. The integrator unit[1] in Figure 2A is similar to the standard ANN model. The rate-coded inputs x_i are multiplied by synaptic weights w_i to produce the membrane

[1]Simple neurons are often approximated as a single integrator units (Torres & Varona, 2012). However, sometimes neurons have multiple sites of integration, i.e. dendritic integration across multiple compartments. We use the name "integrator unit" rather than "neural unit" since a single complex neuron may need to be modeled with multiple integrator units.

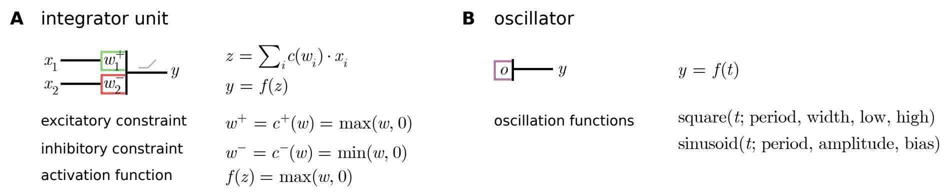


Figure 2: **Architectural Components. A**, Integrator unit for combining signals. Weights represent synaptic connections and are constrained to be either excitatory or inhibitory. Outputs are graded/rate-coded values. **B**, Oscillator for producing driving signals using periodic functions.

potential z, given the resting membrane potential b. A nonlinear activation function $f(z)$ produces the rate-coded output y based on the membrane potential. Unlike the standard ANN model, however, we constrain synaptic weights by a sign constraint function $c(w)$. This is done to reflect that in biological neurons a primary characteristic of a synapse is whether the pre-synaptic input has an excitatory or inhibitory effect on the post-synaptic membrane potential. In the standard model, synapses are initialized with random signs and are free to change during learning. We argue that principled sign constraints are fundamental for interpreting and modeling the logic of neural circuits (Section 3.3), and we show that they are critical for learning in our small, sparse networks (Section 4.2).

Oscillators Neural circuits often feature components with specialized dynamics, with oscillators being a prominent example (Grillner & El Manira, 2020). An oscillator can be implemented through coupled activity between neurons or within a single neuron, similar to pacemaker cells in the heart (Bucher et al., 2015). Oscillators serve as internal drivers of activity, a clear demonstration of the fact that neural circuits do not exclusively react to external inputs from the environment. The simple open-loop oscillator in Figure 2B uses a periodic function $f(t)$ to produce the rate-coded output y. Example periodic functions include square wave and sine wave generators.

3.2 Design Principles

Weight Sharing Animal bodies often feature symmetric and repetitive designs. For example, most animals are bilaterally symmetric, i.e. the left half is virtually identical to the right half. Furthermore, many animals have repeated modules, e.g. centipede legs, snake segments, octopus tentacles. This structure is not only reflected in outward appearance, but also in neural circuitry. Exploiting this structure through weight sharing can improve learning and parameter efficiency (Section 4.2). Weight sharing is also biologically plausible, particularly if the parameter learning process is interpreted not as online *in vivo* learning but rather as evolutionary sculpting of innate, cell-type-specific synaptic densities/excitabilities through genetic mechanisms.

Weight Initialization We initialize synaptic weights with appropriate signs and magnitudes. Magnitudes at or close to zero represent effectively dead synapses and will not produce significant movement. Instead, magnitudes should start off large enough, and the learning algorithm can tune them as necessary. We experiment with both constant and uniform initialization (Section 4.1).

Parameter Learning Architectures designed with the above components are fully differentiable and therefore compatible with backpropagation-based learning algorithms, though not restricted to them. Since lower-level neural circuits governing motor control tend to be small, sparse networks, it is possible to explore directly in parameter space. We experiment with both RL and ES algorithms (Section 4.1).

3.3 Case Study: Swimmer Agent

As a case study, we apply NCAP to translate *C. elegans* locomotion circuits into a simulated Swimmer agent. *C. elegans* is a nematode, i.e. roundworm, that has served as a useful model organism within neuroscience because it is one of the simplest organisms with a nervous system. Moreover, it is unique in that its connectome, i.e. wiring diagram, has been completely mapped (Hall & Altun, 2008). Further, within artificial intelligence, Swimmer agents are common simple models in widely adopted

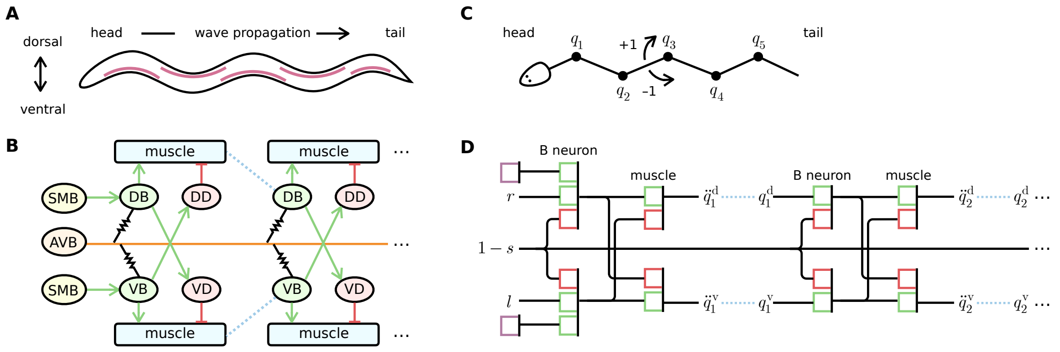


Figure 3: **Swimmer Agent.** **A**, Nematode body. Forward locomotion is achieved through the propagation of muscle waves along the length of the body. **B**, Schematic of neural circuits governing forward locomotion. Excitatory synapse: green line with arrow head. Inhibitory synapse: red line with flat head. Gap junction: resistor symbol. Proprioceptive input: blue dotted line. **C**, Swimmer agent body. Control is achieved through joint accelerations in $[-1, 1]$. **D**, Network architecture for forward locomotion, steering, and stopping.

continuous control benchmarks, e.g. DeepMind Control Suite (Tassa et al., 2020) and OpenAI Gym (Brockman et al., 2016). In the sections below, we first summarize the nematode body structure and neural circuits, and then we use these insights to build our model for the Swimmer.

Nematode Body The nematode body is a 1 mm long, 50 µm diameter tapered cylinder (Figure 3A). It is made up of 959 somatic cells, of which 302 are neurons comprising the nervous system, of which 75 are motor neurons that innervate the 95 body wall muscles distributed along the body. Forward and backward thrust is produced via alternating dorsal-ventral muscle contraction waves propagating down the body in the direction opposite to the direction of motion. Steering is produced by differential activation of the 20 anterior muscles in the head and neck (Gjorgjieva et al., 2014).

Nematode Neural Circuits The nematode forward locomotion circuit is summarized below (Figure 3B). For a more in depth treatment, consider Gjorgjieva et al. (2014); Wen et al. (2018).

Muscle wave propagation is coordinated by 2 classes of neurons that innervate dorsal (D-) and ventral (V-) muscles. B neurons (DB and VB) act as both sensory and motor neurons, expressing stretch receptors in their dendrites to sense bending 200 µm anterior to their somas, and sending excitatory output (via ACh) to the muscles and to D neurons. D neurons (DD and VD) send inhibitory output (via GABA) to the muscles. This microcircuit is modular and repeated down the length of the body, and its logic is interpretable. For a particular module, body bending in the previous module is sensed by B neurons, which then initiate bending on the same side while simultaneously inhibiting bending on the opposite side through D neurons.

Muscle wave initiation is generated by intrinsic oscillators. While proprioception-only circuits (with oscillators ablated) are capable of producing small waves on its own, oscillators are used initiate and entrain larger waves (Gjorgjieva et al., 2014). These oscillators were long believed to only reside in the head and neck, but recently work has shown them to in fact also be present in the body, as it is the B neurons themselves that produce intrinsic oscillations (Wen et al., 2018).

Steering is generated by the differential activation of SMB neurons biasing the head and neck muscles to bend dorsally or ventrally (Izquierdo & Beer, 2015).

Speed control is coordinated by the AVB command neuron, which is connected through gap junctions with all B neurons. When AVB is in a low state, the resting membrane potentials of B neurons are hyperpolarized to prevent activation; when AVB is in a high state, B neurons are free to activate based on proprioceptive and oscillatory inputs.

Swimmer Body The Swimmer agent has an articulated body with N joints connecting $N + 1$ links (Figure 3C). Its movement is entirely within the xy-plane. Thrust is generated by the links pushing against the surrounding fluid, e.g. simulated via a high-Reynolds fluid drag model in DeepMind Control Suite (Tassa et al., 2020; Todorov et al., 2012). The observation space consists of normalized joint positions $\boldsymbol{q} \in [-1, 1]^N$, such that interval boundaries correspond to joint limits. The action space consists of normalized joint accelerations $\ddot{\boldsymbol{q}} \in [-1, 1]^N$, such that interval boundaries correspond to maximum acceleration counterclockwise and clockwise, respectively.

Swimmer Network Architecture The Swimmer agent network architecture is derived by applying NCAP and is best communicated visually (Figure 3D).

For muscle wave propagation, the primary sites of signal integration are B neurons and muscles; D neurons mainly serve to convert opposite-side B neuron outputs from excitatory to inhibitory, and their role can be replicated directly in the muscle integrator units. We model N modules to control each of the N joints. For a particular module $1 \leq i \leq N$, the previous module joint position q_{i-1} is split into dorsal $q_{i-1}^{\text{d}} \in [0, 1]$ and ventral $q_{i-1}^{\text{v}} \in [0, 1]$ components, in order to mirror signals from proprioceptive stretch receptors that are sensitive to bending on one side. B neurons are modeled as integrator units with outputs b_i^{d} and b_i^{v}, which receive same-side excitatory proprioceptive inputs. Muscles are modeled as integrator units with outputs m_i^{d} and m_i^{v}, which receive same-side (ipsilateral) excitatory B neuron input as well as opposite-side (contralateral) inhibitory B neuron input. Finally, the joint acceleration $\ddot{q}_i$ is calculated from dorsal and ventral muscle outputs, which act antagonistically.

For muscle wave initiation, the first module B neurons b_1^{d} and b_1^{v} receive inputs from oscillators o^{d} and o^{v}, respectively, instead of proprioception. We use square wave generators acting in anti-phase.

For steering, SMB outputs are modeled as a right turn signal $r \in [0, 1]$ and a left turn signal $l \in [0, 1]$, which serve as additional excitatory inputs to first module B neurons b_1^{d} and b_1^{v}, respectively.

For speed control, AVB outputs are modeled as a speed signal $s \in [0, 1]$. To approximate the effect of gap junctions, such that $s = 0$ represents stopping and $s = 1$ represents maximum speed, $1 - s$ serves as an additional inhibitory input to all B neurons.

The complete architecture for module i is therefore:

$$\begin{aligned}
q_{i-1}^{\text{d}} &= \max(q_{i-1}, 0) & q_{i-1}^{\text{v}} &= \max(-q_{i-1}, 0) \\
b_i^{\text{d}} &= f\left(w_{\text{prop}}^{+} q_{i-1}^{\text{d}} + w_{\text{speed}}^{-}(1-s)\right) & b_i^{\text{v}} &= f\left(w_{\text{prop}}^{+} q_{i-1}^{\text{v}} + w_{\text{speed}}^{-}(1-s)\right) \\
m_i^{\text{d}} &= f\left(w_{\text{ipsi}}^{+} b_i^{\text{d}} + w_{\text{contra}}^{-} b_i^{\text{v}}\right) & m_i^{\text{v}} &= f\left(w_{\text{ipsi}}^{+} b_i^{\text{v}} + w_{\text{contra}}^{-} b_i^{\text{d}}\right)
\end{aligned}$$

$$\ddot{q}_i = m_i^{\text{d}} - m_i^{\text{v}}$$

with special B neuron integrator units for $i = 1$:

$$b_1^{\text{d}} = f\left(w_{\text{osc}}^{+} o^{\text{d}} + w_{\text{turn}}^{+} r + w_{\text{speed}}^{-}(1-s)\right) \qquad b_1^{\text{v}} = f\left(w_{\text{osc}}^{+} o^{\text{v}} + w_{\text{turn}}^{+} l + w_{\text{speed}}^{-}(1-s)\right)$$

We use weight sharing such that weights with the same name are shared across modules as well as within each module across sides. We initialize all weights with the correct signs and magnitudes of 1.

4 Experiments

Learning Formalization We consider the standard agent-environment interaction model formalized as a Markov Decision Process (MDP). At every timestep t, the agent in state $\boldsymbol{s}_t$ takes an action according to its policy $\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$, receives a reward r_t, and transitions to a new state $\boldsymbol{s}_{t+1}$. Policy parameters $\boldsymbol{\theta}$ are optimized to maximize the discounted return $\sum_{t=0}^{T} \gamma^t r_t$, where T is the horizon of the episode, and $0 < \gamma \leq 1$ is the discount factor.

Learning Algorithms We compare backpropagation-based RL algorithms and a derivative-free ES algorithm for learning parameters in the architecture.

- **Proximal Policy Optimization** (PPO): (Schulman et al., 2017) A model-free, on-policy, policy gradient RL method. It uses a clipped surrogate objective to limit the size of policy change

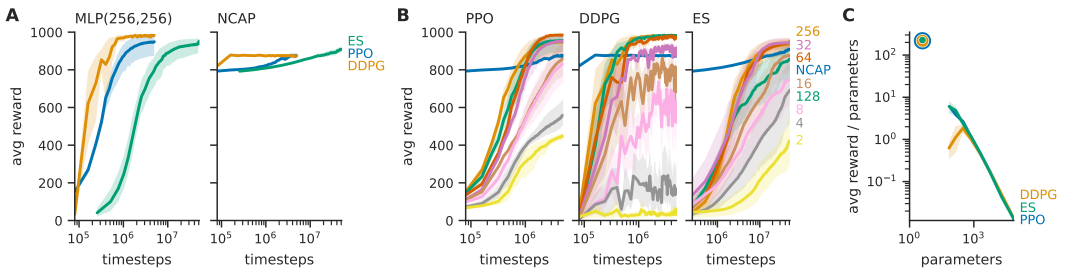


Figure 4: **Performance and Efficiency. A**, Learning curves comparing different algorithms for each architecture. **B**, Learning curves comparing different architectures for each algorithm. **C**, Asymptotic performance divided by parameter count. Solid lines: averages over 10 random seeds. Shaded areas: 95% bootstrap confidence intervals.

at each step, thereby improving stability. Since it assumes stochastic policies, we perturb the deterministic actions with Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

- **Deep Deterministic Policy Gradient** (DDPG): (Lillicrap et al., 2019) A model-free, off-policy, policy gradient RL method. It uses off-policy data and the Bellman equation to learn the Q-function, which is iteratively used to improve the policy.
- **Evolution Strategies** (ES): (Salimans et al., 2017) An evolutionary black-box optimization method. It creates a population of policy parameter variants through perturbations with Gaussian noise, then combines them through averaging, weighted by the return collected across episodes.

Learning Task We implement the Swimmer agent using the 6-link body in the DeepMind Control Suite (Tassa et al., 2020), which is built upon the MuJoCo physics simulator (Todorov et al., 2012). We train the agent on a task with shaped rewards proportional to swimming speed (Appendix A.1).

4.1 Experiment 1: Performance and Efficiency

How well and how efficiently does the NCAP architecture learn compared to MLP architectures?

First, we compare different learning algorithms for either MLP or NCAP architectures (Figure 4A). We use an MLP with 2 hidden layers of dimensions (256, 256) and ReLU nonlinearities. We find that our proposed NCAP architecture achieves comparable asymptotic performance with MLP, while demonstrating higher initial performance. Furthermore, DDPG with NCAP reaches asymptotic performance more data efficiently than with MLP. Using NCAP yields reduced variance during learning between trials (with different random seeds), as well as reduced differences in asymptotic performance between algorithms. For both MLP and NCAP, ES requires roughly an order of magnitude more data to achieve comparable performance with the RL algorithms, consistent with previous work (Salimans et al., 2017). Qualitatively, both MLP and NCAP yield reasonable swimming movement (Videos 1A-B), though NCAP produces waves with large amplitudes resembling *C. elegans* movement, while MLP produces waves with small amplitudes more resembling tadpoles. This different movement shape explains the slightly lower asymptotic performance for NCAP because the body's direction of travel is less correlated with the head orientation, which is relevant for how rewards are calculated (Appendix A.1, Videos 1C).

Second, we compare different architectures for each learning algorithm (Figure 4B). We use MLPs with 2 hidden layers of varying dimension sizes and ReLU nonlinearities. We find that performance deteriorates across all algorithms as MLP hidden dimensions become smaller, with some algorithms like DDPG deteriorating more dramatically. However, even the smallest MLP (hidden dimensions of 2) has many more parameters than the NCAP architecture, with parameter counts of 70 and 4, respectively. Despite an order of magnitude fewer parameters than the smallest MLP, the NCAP architecture starts with high performance and improves to rival the largest MLP (hidden dimensions of 256), with 73,222 parameters. This suggests that the structure of the NCAP architecture provides highly effective inductive biases.

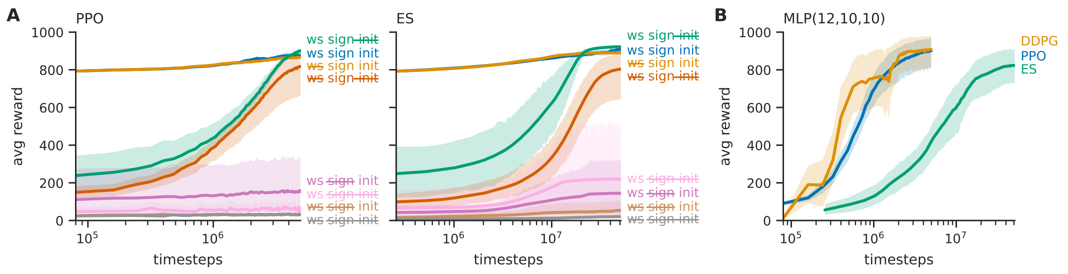


Figure 5: **Ablations.** **A**, Learning curves for different combinations of [weight sharing][sign constraints][initialization]. **B**, Learning curves for equally sized MLP as NCAP.

Third, we compare the asymptotic performance divided by parameter count for each architecture (Figure 4C). This is a measure of parameter efficiency, since for any two architectures with the same asymptotic performance, the one with more parameters will be relatively penalized. We find that, for all algorithms, the NCAP architecture achieves more than an order of magnitude better parameter efficiency than MLPs of any size.

4.2 Experiment 2: Ablations

What are the relative contributions of various NCAP architectural components and design principles?

First, we investigate the role of weight sharing, sign constraints, and weight initialization (Figure 5A). Without weight sharing, weights across modules and across sides are separate parameters, increasing the total number of parameters from 4 to 30. Without sign constraints, the identity function is applied instead of excitatory/inhibitory sign constraints. Without weight initialization, weight magnitudes are initialized through a uniform random distribution within $[0, 1]$, rather than at 1. If using sign constraints, weights are always initialized with the appropriate sign; otherwise, signs are chosen randomly with equal probability. We find that the most significant contribution comes from sign constraints. Without appropriate sign constraints, the NCAP architecture fails to learn during the allotted timesteps, for both RL and ES algorithms. With appropriate sign constraints, even if the weights magnitudes are not initialized ideally, the NCAP architecture will learn. We found that weight sharing has a more minor, but identifiable, contribution to learning.

Second, we investigate the role of sparse connectivity that arises from the natural structure of neural circuits (Figure 5B). Our Swimmer architecture has the convenient special property that it can be completely embedded within an MLP with 3 hidden layers of dimensions (12, 10, 10) and ReLU nonlinearities (Appendix B). Specifically, after ablating sign constraints and weight sharing, our architecture is identical to this MLP with highly pruned connectivity (mostly weights of 0). We remove this sparsity, and we find that the fully connected MLP is able to learn the task with similar asymptotic performance as NCAP. Taken together, our results suggest that principled excitation/inhibition is important in small, sparse networks like our NCAP architecture, but it not as important in large, fully connected networks. This may be related to the "Lottery Ticket Hypothesis" (Frankle & Carbin, 2019), which would suggest that the fully connected, randomly initialized MLPs contain subnetworks ("winning tickets") with initial weights (and signs) that make training particularly effective; these subnetworks are eliminated after once we impose sparsity. More extensive analysis in this direction is left for future work.

5 Discussion

We proposed to use neural circuits to inspire network architectures for embodied control. To this end, we introduced a set of reusable architectural components and design principles, which we applied in a case study that translated *C. elegans* locomotion circuits into a simulated Swimmer agent.

Strengths (1) As we demonstrate in our results, an NCAP architecture can achieve comparable asymptotic performance with standard MLP architectures, while dramatically improving data efficiency and requiring far fewer parameters. This is because, as in animals, the inductive biases provided by the network architecture enable the agent to function well from the start, while continuing to improve with learning. (2) Most complex, learned behavior in animals build upon motor primitives surfaced by low-level spinal neural circuits (Merel et al., 2019a; Rybak et al., 2015). This suggests that architectures derived from neural circuits may in fact be quite general, only constraining the space of learnable behaviors to be naturalistic and not too much more. (3) The local, sparse connectivity of neural circuits can inspire architectures with fewer parameters and that are more interpretable. This facilitates debugging, since network units play more constrained roles. (4) Rather than hand-engineering controllers and equations of motion, which is difficult to do and can produce brittle results, in principle neural circuits provide us with blueprints for building robust, flexible, and transferable solutions that have already been tested in the real-world. As our understanding of neural circuits improves (Braganza & Beck, 2018; Luo, 2021), so too will the quality of our blueprints.

Limitations (1) Domain knowledge of systems neuroscience is needed, or at least very helpful, for deciphering the logic of neural circuits. Literature on neural circuits is largely geared towards readers who are literate in biology and neuroscience, so it can be non-trivial to parse by field outsiders. This is further complicated by current gaps in scientific knowledge, which require modeling assumptions to be made. Fortunately, it is plausible that standard architectures will be designed for widely used agent bodies (e.g. legged robots), which will obviate the need to design architectures from scratch for many use cases. This is similar to the standard practice of reusing architectures like LSTM (Hochreiter & Schmidhuber, 1997) and Transformer (Vaswani et al., 2017), themselves the product of much non-trivial design work. (2) Network architectures derived using NCAP are specific to a body or class of bodies, so they are less transferable, unlike MLPs which are in principle general-purpose. However, this limitation is not solely applicable to architectural priors, since trajectory priors like PMTG (Iscen et al., 2019) make use of body-specific equations of motion. In general, it seems that flexibility is a cost for incorporating such types of priors.

Future Work (1) NCAP can be applied to a wider variety of agent bodies and movements to test its generality. Higher dimensional bodies may exacerbate the need for architectural priors, especially with hierarchical designs. Further, exploring more diverse movements, both rhythmic (e.g. swimming, walking) and discrete (e.g. reaching), may reveal common neural circuit motifs reused within and across animals (Luo, 2021). (2) Different modalities of exteroceptive sensing (e.g. visual, tactile, auditory) can be incorporated into the model. This begins to build up the hierarchy of control systems in animals used to achieve complex, goal-directed behavior. (3) It may be interesting to explore whether large networks (like MLPs) actually learn small network solutions (like NCAP) by analyzing and comparing the learned representations.

Overall, we believe that our work suggests a way of advancing artificial intelligence and robotics research inspired by systems neuroscience. Encouragingly, recent developments have unlocked the ability to map and manipulate neural circuits with a precision never before possible, providing an exciting new source of insights. Conversely, embodied agents can be a fruitful testbed for mechanistic models of neural circuits, and *in silico* results can suggest areas of further neuroscientific study. We hope that this will perpetuate a "virtuous cycle" in which the science and engineering of intelligence may progress together (Hassabis et al., 2017).

REFERENCES

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Kohd, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal

Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv:2108.07258 [cs]*, August 2021.

Jordan H. Boyle, Stefano Berri, and Netta Cohen. Gait modulation in C. elegans: An integrated neuromechanical model. *Frontiers in Computational Neuroscience*, 6, 2012.

Oliver Braganza and Heinz Beck. The circuit motif as a conceptual tool for multilevel neuroscience. *Trends in Neurosciences*, 41(3):128–136, March 2018.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

Dirk Bucher, Gal Haspel, Jorge Golowasch, and Farzan Nadim. Central pattern generators. November 2015.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv:1808.05377 [cs, stat]*, April 2019.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding sparse, trainable neural networks. *arXiv:1803.03635 [cs]*, March 2019.

Julijana Gjorgjieva, David Biron, and Gal Haspel. Neurobiology of Caenorhabditis elegans locomotion: Where do we stand? *BioScience*, 64(6):476–486, June 2014.

Sten Grillner and Abdeljabbar El Manira. Current principles of motor control, with special reference to vertebrate locomotion. *Physiological Reviews*, 100(1):271–320, January 2020.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *arXiv:1512.07108 [cs]*, October 2017.

David H. Hall and Zeynep F. Altun. *C. Elegans Atlas*. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, N.Y, 2008.

Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, July 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385 [cs]*, December 2015.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv:1610.05182 [cs]*, October 2016.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *arXiv:1707.02286 [cs]*, July 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, November 1997.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning; lessons we've learned. *The International Journal of Robotics Research*, 40(4-5):698–721, April 2021.

Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Policies modulating trajectory generators. *arXiv:1910.02812 [cs]*, October 2019.

Eduardo J. Izquierdo and Randall D. Beer. An integrated neuromechanical model of steering in C. elegans. In *07/20/2015-07/24/2015*, pp. 199–206. The MIT Press, July 2015.

Goktug Karakasli. ESTorch: An evolution strategy library built around PyTorch, 2020.

O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, February 1987.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv:1907.03146 [cs]*, November 2020.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, December 1989.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv:1504.00702 [cs]*, April 2016.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019.

Grace W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, 33(10):2017–2031, September 2021.

Liqun Luo. Architectures of neuronal circuits. *Science*, 373(6559), September 2021.

Josh Merel, Matthew Botvinick, and Greg Wayne. Hierarchical motor control in mammals and machines. *Nature Communications*, 10(1):5489, December 2019a.

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. *arXiv:1811.11711 [cs]*, January 2019b.

Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *arXiv:2011.07537 [cs]*, May 2021.

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pp. 763–768, Kobe, May 2009. IEEE.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *arXiv:1912.01703 [cs, stat]*, December 2019.

Harry A. Pierson and Michael S. Gashler. Deep learning in robotics: A review of recent research. *Advanced Robotics*, 31(16):821–835, August 2017.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *arXiv:2003.13678 [cs]*, March 2020.

Ilya A. Rybak, Kimberly J. Dougherty, and Natalia A. Shevtsova. Organization of the mammalian locomotor CPG: Review of computational model and circuit architectures based on genetically identified spinal interneurons. *eneuro*, 2(5):ENEURO.0069–15.2015, September 2015.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864 [cs, stat]*, September 2017.

Gopal P. Sarma, Chee Wai Lee, Tom Portegys, Vahid Ghayoomie, Travis Jacobs, Bradly Alicea, Matteo Cantarelli, Michael Currie, Richard C. Gerkin, Shane Gingell, Padraig Gleeson, Richard Gordon, Ramin M. Hasani, Giovanni Idili, Sergey Khayrulin, David Lung, Andrey Palyanov, Mark Watts, and Stephen D. Larson. OpenWorm: Overview and recent advances in integrative biological simulation of Caenorhabditis elegans. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):20170382, October 2018.

Stefan Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. In Hiroshi Kimura, Kazuo Tsuchiya, Akio Ishiguro, and Hartmut Witte (eds.), *Adaptive Motion of Animals and Machines*, pp. 261–280. Springer-Verlag, Tokyo, 2006.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs]*, August 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556 [cs]*, April 2015.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Piotr Trochim, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. Dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, November 2020.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Vilamoura-Algarve, Portugal, October 2012. IEEE.

Joaquin J. Torres and Pablo Varona. Modeling biological neural networks. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok (eds.), *Handbook of Natural Computing*, pp. 533–564. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv:1706.03762 [cs]*, December 2017.

Quan Wen, Shangbang Gao, and Mei Zhen. Caenorhabditis elegans excitatory ventral cord motor neurons derive rhythm for body undulation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):20170370, October 2018.

Anthony M. Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature Communications*, 10(1):3770, December 2019.

A Experimental Details

A.1 Tasks

The `swim` task aims to test the agent's ability to swim forwards at a desired speed. It returns a smooth reward that is 0 when stopped or moving backwards, and rises linearly to and saturates at 1 when swimming at the desired speed.

A.2 Implementation

Libraries Neural networks were implemented in PyTorch (Paszke et al., 2019). The RL algorithms were implemented using Tonic (Pardo, 2021). The ES algorithm was implemented using ES Torch (Karakasli, 2020).

Computational Resources Training was performed on a high performance computing cluster running the Linux Ubuntu operatin system. RL algorithm training runs were parallelized over 8 cores, while ES algorithm runs were parallelized over 32 cores.

A.3 Hyperparameters

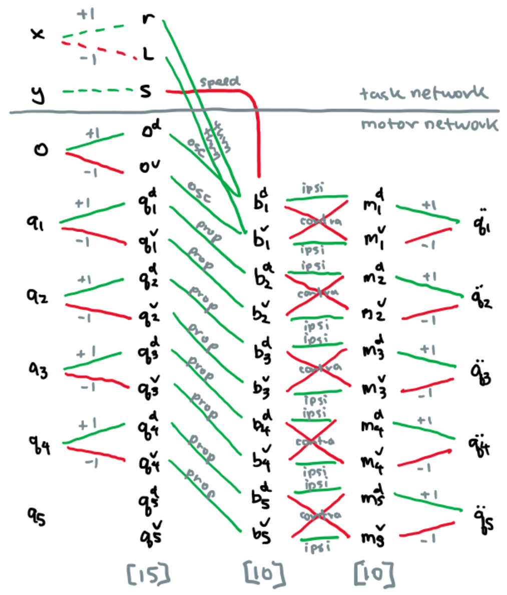
RL Algorithms Standard hyperparameters for PPO and DDPG in Tonic (Pardo, 2021) at commit `48a7b72`; timesteps, 5e6.

ES Algorithm Population size, 256; noise standard deviation σ, 0.02; L2 weight decay, 0.005; optimized, Adam; learning rate, 0.01; timesteps, 5e7.

NCAP Swimmer Oscillator, square wave with period of 60 timesteps.

B Swimmer Architecture Details

The NCAP architecture for Swimmer can be embedded within a fully connected MLP of 3 hidden layers.


task network
motor network
speed
turn
osc
prop
ipsi
contra
+1
-1
[15]
[10]
[10]