Neural Circuit Architectural Priors for Embodied Control

Nikhil X. Bhattasali, Anthony M. Zador, Tatiana A. Engel
NeuroAI Program, Cold Spring Harbor Laboratory
{bhattas,zador,engel}@cshl.edu

Abstract

Artificial neural networks for motor control usually adopt generic architectures like fully connected MLPs. While general, these *tabula rasa* architectures rely on large amounts of experience to learn, are not easily transferable to new bodies, and have internal dynamics that are difficult to interpret. In nature, animals are born with highly structured connectivity in their nervous systems shaped by evolution; this innate circuitry acts synergistically with learning mechanisms to provide inductive biases that enable most animals to function well soon after birth and learn efficiently. Convolutional networks inspired by visual circuitry have encoded useful biases for vision. However, it is unknown the extent to which ANN architectures inspired by neural circuitry can yield useful biases for other AI domains. In this work, we ask what advantages biologically inspired ANN architecture can provide in the domain of motor control. Specifically, we translate *C. elegans* locomotion circuits into an ANN model controlling a simulated Swimmer agent. On a locomotion task, our architecture achieves good initial performance and asymptotic performance comparable with MLPs, while dramatically improving data efficiency and requiring orders of magnitude fewer parameters. Our architecture is interpretable and transfers to new body designs. An ablation analysis shows that constrained excitation/inhibition is crucial for learning, while weight initialization contributes to good initial performance. Our work demonstrates several advantages of biologically inspired ANN architecture and encourages future work in more complex embodied control.

1 Introduction

Artificial neural networks (ANNs) for motor control usually adopt generic architectures like fully connected multi-layered perceptrons (MLPs) [Pierson and Gashler, 2017, Levine et al., 2016, Bin Peng et al., 2020, Heess et al., 2016]. While general, these *tabula rasa* architectures rely on large amounts of experience to learn. Data efficiency is especially desirable because, unlike computer vision and natural language processing which have benefited greatly from available large datasets [Bommasani et al., 2021], motor control requires interacting with an environment to gather experience, which is time-consuming, laborious, and possibly unsafe [Kroemer et al., 2020]. Transfer is also a challenge, as experience is usually body-specific, and an ANN trained to control one body is not easily adapted to new bodies. In addition, architectures like MLPs are difficult to interpret, as their internal dynamics are distributed across units and non-trivial to relate to agent behavior [Merel et al., 2019a].

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]. 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 learning mechanisms act synergistically, enabling most animals to function well soon after birth, while continuing to acquire and improve skills efficiently, e.g. a horse learning to walk with only a couple hours of experience. Moreover, despite

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

species-specific variations, there is a significant amount of shared architecture (e.g. cerebellum, basal ganglia) and design principles (e.g. hierarchical modularity, partial autonomy) even between distantly related species [Merel et al., 2019b]. The connectivity of neural circuits is often highly structured and sparse [Luo, 2021], in sharp contrast to the all-to-all connectivity of MLPs. Moreover, evolution has progressively built more advanced abilities on lower-level circuits, leveraging modular structure to transfer existing working designs to new animal bodies [Cisek, 2019, Merel et al., 2019b]. Taken together, these findings from neuroscience suggest that structured neural circuits in animals instantiate efficient, transferable, and modular solutions for high-dimensional embodied control.

Capturing some of this structure in model architecture may enable ANNs to narrow the gap between artificial and natural systems. For example, convolutional networks were inspired by biological visual circuitry, and the inductive biases of spatial locality and weight sharing that they encode have yielded substantial improvements in performance, data efficiency, and parameter efficiency for vision tasks [Lindsay, 2021]. However, it is unknown the extent to which ANN architectures inspired by neural circuitry can yield useful inductive biases for other AI domains.

In this work, we ask what advantages biologically inspired ANN architecture can provide in the domain of motor control. Specifically, we translate *C. elegans* locomotion circuits into an ANN model controlling a simulated Swimmer agent selected from a standard AI benchmark [Tassa et al., 2020]. Our architecture is an instance of what we call a "Neural Circuit Architectural Prior" (NCAP), to denote an ANN architecture that encodes prior structure inspired by biological neural circuits.

On a locomotion task, our architecture achieves good initial performance and asymptotic performance comparable with MLPs, while dramatically improving data efficiency and requiring orders of magnitude fewer parameters. Our architecture is interpretable and transfers to new body designs. An ablation analysis shows that constrained excitation/inhibition is crucial for learning, while weight initialization contributes to good initial performance. Our work demonstrates several advantages of biologically inspired ANN architecture and encourages future work in more complex embodied control.

In summary, the primary contributions of this work are:

1. An ANN architecture inspired by *C. elegans* locomotion circuits that combines the discrete-time ANN formalism that is standard in machine learning with features from computational neuroscience like constraints on synapse sign (i.e. excitation vs. inhibition) and special cell types (i.e. intrinsic oscillators);
2. An evaluation of our model's initial performance, asymptotic performance, data efficiency, parameter efficiency, interpretability, and transfer compared to standard MLP architectures; and
3. An ablation analysis of the effects of weight sharing, sign constraints, initialization, and sparse connectivity on performance and learning.

Code and videos are available at: `https://sites.google.com/view/ncap-swimmer`

2 Related Work

Movement Priors A motor controller ultimately generates joint-space torques $\boldsymbol{\tau}$ to apply at each actuator. While a learning-based controller can directly generate torques [Levine et al., 2016] or world-space positions $\boldsymbol{x}$ or accelerations $\ddot{\boldsymbol{x}}$ that are transformed into torques [Khatib, 1987], movement priors are often adopted to introduce abstraction, incorporate prior knowledge, and improve performance and learning speed [Kroemer et al., 2020]. Movement priors can be usefully grouped into 3 categories:

(1) *Trajectory Priors* encode desired movement through analytic equations of motion, e.g. Dynamic Movement Primitives use parameterized differential equations to implement stable attractor dynamical systems expressing different motion shapes [Schaal, 2006, Pastor et al., 2009], and Policies Modulating Trajectory Generators augment hand-engineered trajectories with learned residual terms to enhance robustness and flexibility [Iscen et al., 2019]. Generally, trajectory priors are interpretable and work well when desired movement is adequately captured by analytic primitives (e.g. sinusoids, splines); however, they can require much time and manual effort to design robustly.

(2) *Behavioral Imitation Priors* encode desired movement by training parameterized functions like ANNs to imitate reference motions generated from motion capture or manual keyframing, e.g. Bin Peng et al. [2020] train expert policies to imitate animal motion capture, and Neural

Probabilistic Motor Primitives compress expert policies trained to imitate human motion capture into a common, reusable embedding space [Merel et al., 2019c]. Generally, behavioral imitation priors have reproduced diverse movements and trade the time/effort of designing equations with that of compiling reference motions from humans/animals and retargeting the motions to agent bodies.

(3) *Architectural Priors* encode desired movement indirectly through specialized ANN architectures that establish inductive biases, e.g. Heess et al. [2016] decompose an agent into a loosely neuro-inspired hierarchy consisting of a high-level "cortical network" with access to exteroceptive signals and a low-level "spinal network" with access to only proprioceptive signals, and Heess et al. [2017] extend this architecture to more challenging locomotion tasks. Generally, architectural priors can significantly improve performance and data efficiency over generic/monolithic architectures; however, architectural priors still represent an under-explored space in the AI motor control literature, and much existing work has only specified high-level architectural structure while retaining generic low-level connectivity, e.g. the above-referenced "cortical" and "spinal" networks both use dense MLP connectivity, rather than connectivity that resembles biological cortical and spinal circuitry.

In this work, we investigate the potential of architectural priors that more closely resemble neural circuitry. In doing so, we demonstrate one path for how prior knowledge can be encoded in AI agents and how neuroscience can inspire more naturalistic AI [Hassabis et al., 2017].

Central Pattern Generators In bio-inspired motor control, an extensive literature has built upon models of Central Pattern Generators (CPGs), which are neural circuits found across animal species that produce rhythmic activity in the absence of rhythmic inputs and sensory feedback. These circuits underlie fundamental rhythmic movements, including breathing, chewing, digesting, swimming, walking, and running. Computational models of CPGs have been successfully applied to diverse robotic bodies. For a review, refer to Ijspeert [2008] and Yu et al. [2014].

While our concept of a Neural Circuit Architectural Prior (NCAP) is related to CPGs, it differs in modeling constraints and formalization: (1) Neural-circuit-inspired connectivity can generate movement that includes, but is not limited to, rhythmic movement. In animals, of course, discrete movements (e.g. reflexes, reaching, sitting, jumping) are generated by neural circuits too. This work highlights the case study of swimming because *C. elegans* locomotion circuits are simple to explain, but related works have also explored circuits for reflexes [Liu et al., 2018], reaching [Schaal and Schweighofer, 2005], and decision making [Lechner et al., 2019, Hasani et al., 2020]. (2) CPG models vary significantly, with some based on neural networks (i.e. architectural priors) and many based on other formalisms including systems of coupled oscillators, vector maps, and finite-state machines [Ijspeert, 2008]. Neural-network-based CPGs further vary between the biophysically detailed, spiking, rate-coded, and population-level [Torres and Varona, 2012, Bing et al., 2018, Ijspeert, 2008]. This work adopts a discrete-time ANN formalism that is familiar to the broader AI community. In addition to pedagogical benefits, our formalism produces models that are fully differentiable, enabling us to tune parameters with the same reinforcement learning (RL) and evolution strategies (ES) algorithms commonly used in AI motor control. Further, our Swimmer NCAP architecture has the special property of being embeddable within a standard MLP of certain dimensions (Appendix B), enabling a direct, rigorous, and novel investigation of low-level connectivity in ANNs.

Neuromechanical Models In computational neuroscience, models of neural circuits have been increasingly combined with models of biomechanics to study brain-body-environment interactions [Ausborn et al., 2021, Rybak et al., 2015, Danner et al., 2017]. Our work builds upon the rich insights gained from neuromechanical modeling, in particular of *C. elegans* swimming and steering circuits [Boyle et al., 2012, Demin and Vityaev, 2014, Izquierdo and Beer, 2015, Sarma et al., 2018]. However, while neuromechanical work primarily aims to elucidate biological principles and explain neural/behavioral data, our work has the distinct goal of translating insights to AI. To this end, we adopt the discrete-time ANN formalism as described above, and we also target a body from a standard AI benchmark: the DeepMind Control Suite [Tassa et al., 2020] built on the MuJoCo physics simulator [Todorov et al., 2012]. In contrast, the above-mentioned *C. elegans* works use biophysically realistic bodies and muscles. Our Swimmer body is significantly different from *C. elegans* in mechanics, degrees of freedom, and actuators; it is not obvious that *C. elegans* circuits would be useful, and our findings may encourage future work for other MuJoCo bodies as well. Ultimately, we hope our work can strengthen the bridge between the neuromechanical modeling and AI communities.

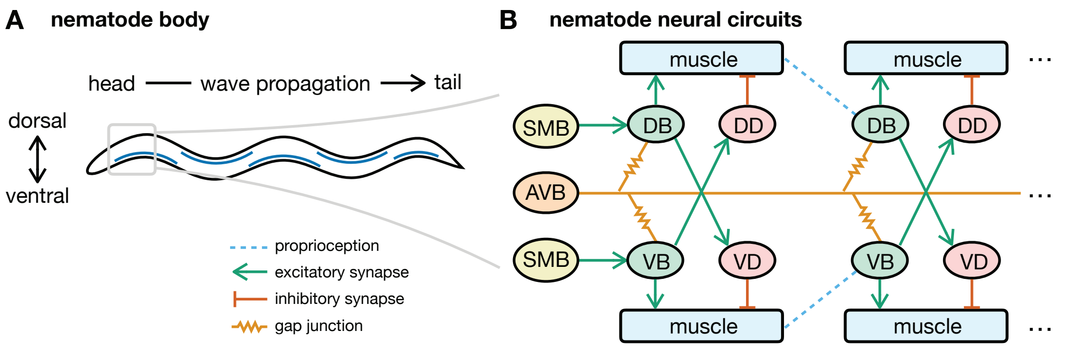


Figure 1: **Nematode. (A)** The nematode achieves forward locomotion through dorsal-ventral muscle contraction waves propagating down the body. **(B)** Muscle wave propagation, oscillation, steering, and speed control are coordinated by a highly stereotyped, modular, and repeated microcircuit. B neurons sense bending in the previous module and excite ipsilateral muscles, while inhibiting contralateral muscles via D neurons. Intrinsic oscillations in B neurons initiate waves. SMB neurons bias head/neck muscles for steering. AVB attenuates all B neurons via gap junctions for speed control.

3 Model

We translate nematode (*C. elegans*) locomotion circuits into an ANN model controlling a simulated Swimmer agent. In Section 3.1, we provide an overview of the nematode body and the neural circuitry underlying locomotion. In Section 3.2, we describe the integrator and oscillator units that serve as building blocks for our NCAP architecture. In Section 3.3, we formalize the observation and action space of the Swimmer agent, and we propose our NCAP architecture inspired by nematode circuits.

3.1 Nematode

The nematode *C. elegans* 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 and Altun, 2008].

Nematode Body The nematode body is a 1 mm long, 50 μm diameter tapered cylinder (Figure 1A). 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 1B). For an in-depth treatment, refer to Gjorgjieva et al. [2014] and 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 highly stereotyped, 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 (ipsilateral) while simultaneously inhibiting bending on the opposite side (contralateral) 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 there is intrinsic oscillatory activity within the B neurons themselves [Wen et al., 2018].

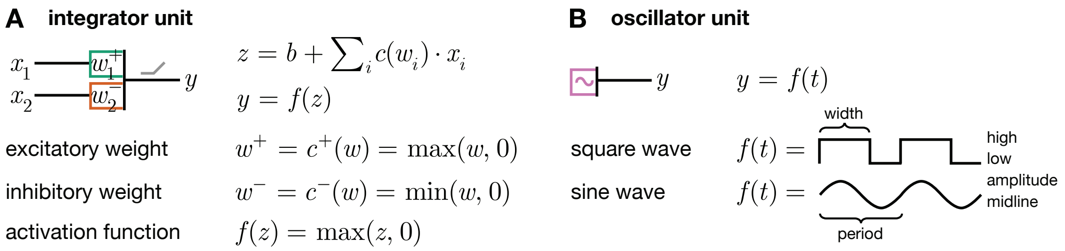

A integrator unit
x_1
x_2
w_1^+
w_2^-
y
$z = b + \sum_i c(w_i) \cdot x_i$
$y = f(z)$
excitatory weight $w^+ = c^+(w) = \max(w, 0)$
inhibitory weight $w^- = c^-(w) = \min(w, 0)$
activation function $f(z) = \max(z, 0)$
B oscillator unit
y
$y = f(t)$
width
square wave $f(t) =$
high
low
sine wave $f(t) =$
amplitude
midline
period

Figure 2: **Architectural Components. (A)** An integrator unit models a simple neuron. The graded input signals are multiplied by weights that represent synaptic efficacy and that are constrained to be either positive (excitatory, green boxes) or negative (inhibitory, red boxes). The graded output signal is produced by an activation function. **(B)** An oscillator unit produces driving signals much like intrinsic pacemaker cells and network-based oscillators. The graded output signal is generated by periodic functions, e.g. square waves and sine waves.

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

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

3.2 Architectural Components

Our NCAP architecture is built from components that combine the discrete-time ANN formalism that is standard in machine learning with features from computational neuroscience like constraints on synapse sign (i.e. excitation vs. inhibition) and special cell types (i.e. intrinsic oscillator). The components are fully differentiable and therefore compatible with backpropagation-based learning algorithms, though not restricted to them.

Integrator Units Signals in biological neural circuits are processed and integrated by neurons. The integrator unit[1] in Figure 2A is similar to the standard ANN model. The graded inputs x_i are multiplied by synaptic weights w_i to produce the membrane potential z, given the resting membrane potential b. A nonlinear activation function $f(z)$ produces the graded 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 circuits a primary characteristic of a synapse is whether it is excitatory or inhibitory. In the standard model, synapses are initialized with random signs and are free to change during learning. We argue that constrained excitation/inhibition is fundamental for interpreting and modeling the logic of neural circuits, and we show in the ablation analysis that they are critical for learning in our architecture (Section 4.5).

Oscillator Units Neural circuits often feature components with specialized dynamics, with oscillators being a prominent example [Grillner and 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, exemplifying the fact (less appreciated within the ML community) that neural circuits are not exclusively driven by external inputs from the environment. The oscillator unit in Figure 2B uses a periodic function $f(t)$ to produce the graded output y. Example periodic functions include square wave and sine wave generators.

3.3 Swimmer

The Swimmer is an agent body in widely adopted AI motor control benchmarks, e.g. DeepMind Control Suite [Tassa et al., 2020] and OpenAI Gym [Brockman et al., 2016]. We target this standard body rather than a biorealistic body like previous work [Sarma et al., 2018, Izquierdo and Beer, 2015]

[1]Simple neurons are often approximated as a single integrator units [Torres and Varona, 2012]. However, sometimes neurons have multiple sites of integration, i.e. dendritic integration across multiple compartments. We prefer "integrator unit" to"neural unit" as a complex neuron may require multiple integrator units to model.

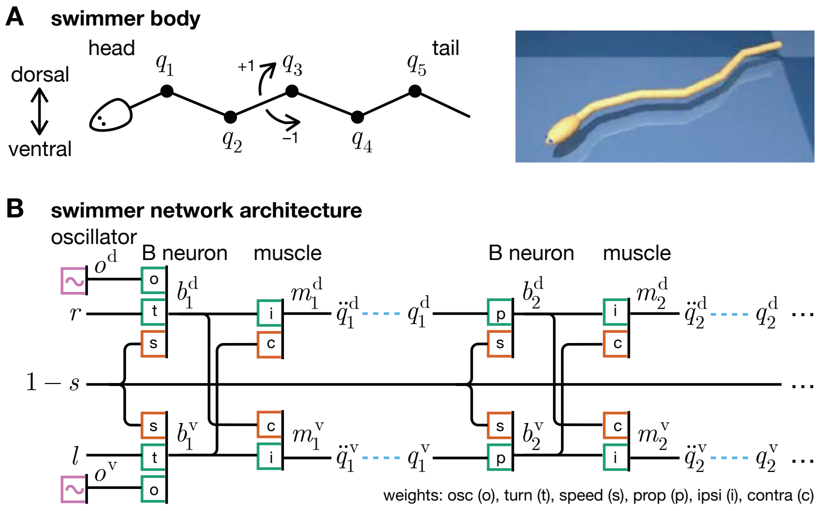

A swimmer body
head
tail
dorsal
ventral
q_1 q_2 q_3 q_4 q_5
+1
−1
B swimmer network architecture
oscillator
B neuron
muscle
o^d o^v r l $1-s$
b_1^d b_1^v m_1^d m_1^v $\ddot{q}_1^d$ $\ddot{q}_1^v$ q_1^d q_1^v
b_2^d b_2^v m_2^d m_2^v $\ddot{q}_2^d$ $\ddot{q}_2^v$ q_2^d q_2^v
weights: osc (o), turn (t), speed (s), prop (p), ipsi (i), contra (c)

Figure 3: **Swimmer. (A)** The Swimmer has an articulated body with N joints connecting $N+1$ links ($N=5$ shown). Its observation space is normalized joint positions $\boldsymbol{q}$, and its action space is normalized joint accelerations $\ddot{\boldsymbol{q}}$. **(B)** Our network architecture closely conforms to the modular microcircuit of the nematode. Each module i senses bending in the previous module q_{i-1} and drives B neurons b_i and muscles m_i, which are combined to create joint accelerations $\ddot{q}_i$.

in order to investigate the potential of biologically inspired network architecture in agents used by the broader AI community and to inspire future work in other MuJoCo bodies as well.

Swimmer Body The Swimmer agent has an articulated body with N joints connecting $N+1$ links (Figure 3A). 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 [Todorov et al., 2012]. The observation space consists of normalized joint positions $\boldsymbol{q} \in [-1,1]^N$ between joint limits. The action space consists of normalized joint accelerations $\ddot{\boldsymbol{q}} \in [-1,1]^N$ between maximum acceleration counterclockwise and clockwise, respectively.

Swimmer Network Architecture Our NCAP architecture is best explained visually (Figure 3B).

For muscle wave propagation, signals are integrated in B neurons and muscles; D neurons mainly serve to convert opposite-side B neuron signals 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 $1 \le i \le N$, the previous module joint position q_{i-1} is split into dorsal $q_{i-1}^{\mathrm{d}} \in [0,1]$ and ventral $q_{i-1}^{\mathrm{v}} \in [0,1]$ 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:

$$
\begin{aligned}
q^{\mathrm{d}}_{i-1} &= \max(q_{i-1}, 0) & q^{\mathrm{v}}_{i-1} &= \max(-q_{i-1}, 0) \\
b^{\mathrm{d}}_{i} &= f\left(w^{+}_{\text{prop}} q^{\mathrm{d}}_{i-1} + w^{-}_{\text{speed}}(1-s)\right) & b^{\mathrm{v}}_{i} &= f\left(w^{+}_{\text{prop}} q^{\mathrm{v}}_{i-1} + w^{-}_{\text{speed}}(1-s)\right) \\
m^{\mathrm{d}}_{i} &= f\left(w^{+}_{\text{ipsi}} b^{\mathrm{d}}_{i} + w^{-}_{\text{contra}} b^{\mathrm{v}}_{i}\right) & m^{\mathrm{v}}_{i} &= f\left(w^{+}_{\text{ipsi}} b^{\mathrm{v}}_{i} + w^{-}_{\text{contra}} b^{\mathrm{d}}_{i}\right)
\end{aligned}
$$

$$\ddot{q}_i = \min(m^{\mathrm{d}}_i, 1) - \min(m^{\mathrm{v}}_i, 1)$$

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

$$b^{\mathrm{d}}_1 = f\left(w^{+}_{\text{osc}} o^{\mathrm{d}} + w^{+}_{\text{turn}} r + w^{-}_{\text{speed}}(1-s)\right) \qquad b^{\mathrm{v}}_1 = f\left(w^{+}_{\text{osc}} o^{\mathrm{v}} + w^{+}_{\text{turn}} l + w^{-}_{\text{speed}}(1-s)\right)$$

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 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 $\boldsymbol{\theta}$ with our architecture described in Section 3.3. 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 Setup We implement the Swimmer using the standard $N = 5$ body in the DeepMind Control Suite [Tassa et al., 2020] built upon the MuJoCo physics simulator [Todorov et al., 2012]. We train the agent to swim using shaped rewards proportional to swimming speed (Appendix A.1).

Learning Algorithms We compare backpropagation-based RL algorithms (PPO, DDPG) and a derivative-free ES algorithm (OpenAI-ES) for tuning parameters in the architecture (Appendix A.2).

4.1 Performance and Data Efficiency

How well and data efficiently does learning occur in NCAP vs. MLP architectures?

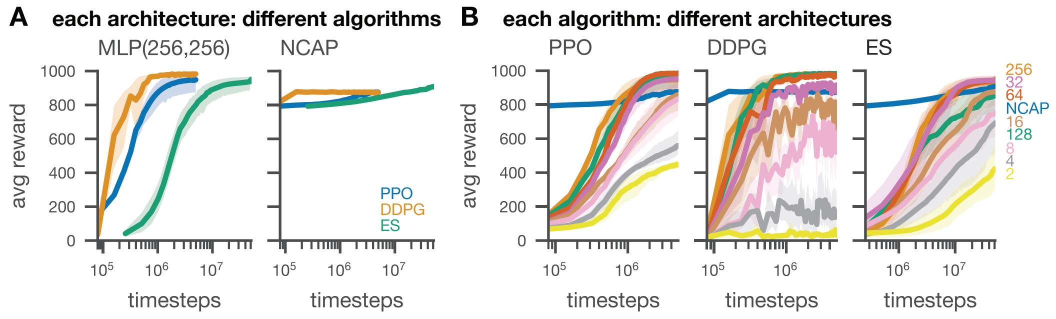


Figure 4: **Performance and Data Efficiency. (A)** Comparison across architectures, using different algorithms. Our architecture starts with high reward and improves with learning, achieving significantly better data efficiency and comparable performance. **(B)** Comparison across algorithms, using different architectures. Our architecture with 4 parameters overperforms small MLPs (e.g. MLP(2,2) has 74 parameters) and is comparable to large MLPs (e.g. MLP(256, 256) has 73,226 parameters). Therefore, structured connectivity (not simply parameter count) matters for performance. Plots show averages over 10 random seeds (solid lines) and 95% bootstrap confidence intervals (shaded areas).

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 NCAP architecture achieves substantially higher initial performance than MLPs as well as comparable asymptotic performance with MLPs, demonstrating the effectiveness of prior knowledge encoded in network architecture. Our NCAP architecture shows 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 architectures 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). We note that we simplified our design from actual *C. elegans* circuits for pedagogical reasons (Section 4.5), and our goal is not to solve this swimming task *per se* but rather to investigate the advantages of biologically inspired network architecture more generally. *C. elegans* circuits are not optimized for fast swimming with few segments (Section 4.4); future work may propose architectures better for this specific task, e.g. using larval zebrafish circuits.

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 for MLPs as hidden dimensions become smaller, with some algorithms like DDPG deteriorating dramatically. However, our NCAP architecture with 4 parameters overperforms small MLPs and is comparable to large MLPs. This is especially notable as the smallest MLP(2,2) has 74 parameters (1 order of magnitude more than NCAP) and the largest MLP(256,256) has 73,226 parameters (2 orders of magnitude more than NCAP). This suggests that the relatively simple structure of our NCAP architecture provides highly effective inductive biases.

4.2 Parameter Efficiency

How valuable are parameters in NCAP vs. MLP architectures?

We compare the asymptotic performance divided by parameter count for each architecture (Figure 5). Across all algorithms, our NCAP architecture achieves more than an order of magnitude better parameter efficiency than MLPs of any size. A parameter in our NCAP architecture is "worth more" than one in MLPs.

parameter count & efficiency

model	params	$\log_2$(reward / params)
(256,256)	73,226	-6.3 ± 0.2
(128,128)	20,234	-4.4 ± 0.2
(64,64)	6,026	-2.6 ± 0.1
(32,32)	1,994	-1.1 ± 0.2
(16,16)	746	0.2 ± 0.3
(8,8)	314	1.4 ± 0.4
(4,4)	146	1.9 ± 0.3
(2,2)	74	2.6 ± 0.2
NCAP	**4**	**7.8 ± 0.1**

Figure 5:**Parameter Efficiency.** Asymptotic reward per unit parameter. Our model achieves better parameter efficiency than MLPs of all tested sizes.

4.3 Interpretability

How interpretable are unit dynamics in our NCAP architecture?

We visualize network unit dynamics during a task (Videos 2). Since our NCAP architecture is a sparsely connected, modular network where units play constrained roles (e.g. excitatory or inhibitory), it is easier to relate unit dynamics to agent behavior, which facilitates debugging and engineering constraints for safety.

4.4 Transfer

How well does our NCAP architecture adapt to new bodies?

We leverage the modular structure of our NCAP architecture to adapt a trained network for $N = 5$ joints to bodies with different N' joints by adding/removing modules (Figure 6). Even without further training, we achieve robust swimming for a wide range of N' (Videos 3). Interestingly, performance is *better* for longer bodies, possibly reflecting that nematode circuits are evolved for a more segmented body. This kind of zero-shot transfer is not possible with monolithic, densely connected MLP architectures.

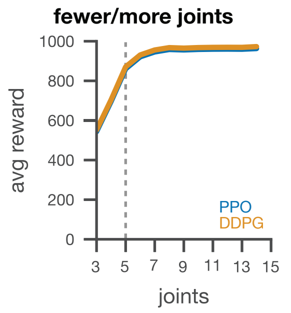


Figure 6: **Transfer.** After training with $N = 5$ joints, our model can transfer to different N' by adding/removing modules.

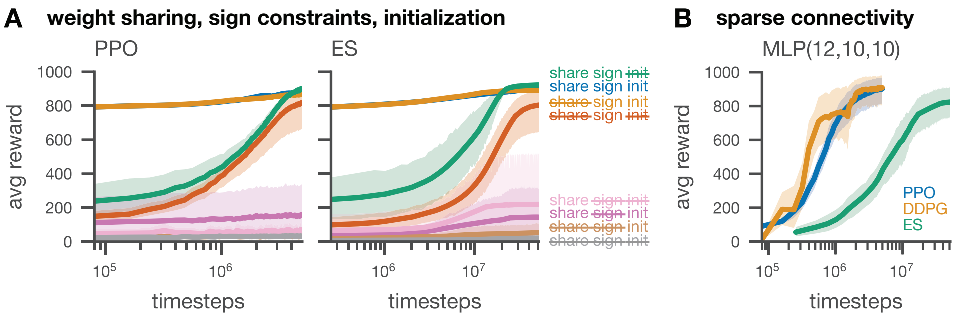


Figure 7: **Ablations. (A)** Ablations of weight sharing, sign constraints, and initialization in different combinations. Sign constraints are crucial for learning. Weight initialization contributes to good initial performance. Weight sharing yields a small gain. **(B)** Ablation of sparse connectivity yields an equivalently sized MLP (Appendix B). Learning is restored without sparse connectivity.

4.5 Ablations

What are the effects of various features of our NCAP architecture on performance and learning?

First, we investigate the role of weight sharing, sign constraints, and weight initialization (Figure 7A). 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 to weights instead of the constraint function. Without typical 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 sign constraints are crucial for learning. Without appropriate sign constraints, our 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 large, our NCAP architecture will learn. Weight initialization is responsible for good initial performance. Weight sharing has a smaller, but identifiable, contribution to data efficiency.

Second, we investigate the role of sparse connectivity that arises from the natural structure of neural circuits (Figure 7B). Our Swimmer architecture has the 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 find that the MLP can learn the task with similar asymptotic performance as our NCAP.

Taken together, our results suggest that constrained excitation/inhibition is an important consideration in small, sparse architectures like our NCAP, but less important in MLPs. This may be related to the "Lottery Ticket Hypothesis" [Frankle and Carbin, 2019], which suggests that, upon initialization, the MLPs already contain subnetworks with initial weights *and signs* that do most of the work for learning, i.e. they are "winning tickets"; imposing sparsity eliminates these overlapping subnetworks.

5 Discussion

We asked what advantages biologically inspired ANN architecture can provide in the domain of motor control. Through our case study translating *C. elegans* locomotion circuits into an ANN model, we found that biologically inspired ANN architecture can achieve comparable asymptotic performance to MLPs with significantly improved initial performance, data efficiency, parameter efficiency, interpretability, and transfer. Therefore, while *tabula rasa* architectures are general, Neural Circuit Architectural Priors (NCAP) can provide useful inductive biases for motor control.

Nevertheless, our case study here was limited: it focused on a relatively simple body and circuit, and it prioritized pedagogical simplicity over performance optimization and behavioral complexity. Future work should explore additional bodies, movement types, and circuit modalities:

(1) *Bodies*. Many animals generate movement through highly structured pattern generator circuits, including quadrupeds (e.g. mice, cats, dogs, horses) and bipeds (e.g. birds, humans) [Ijspeert, 2008].

In particular, quadrupeds are commonly studied in both neuroscience and robotics [Rybak et al., 2015, Danner et al., 2017, Iscen et al., 2019]. Quadruped locomotion circuits are located primarily within the spinal cord and produce robust gaits (e.g. walk, bound, gallop) even when top-down connections from the brain are lesioned [Buschmann et al., 2015]. Insights from these circuits could be translated to AI motor control bodies, and a Quadruped NCAP architecture may use similar integrators, oscillators, and constrained connections as our Swimmer NCAP.

(2) *Movement Types*. Neural circuits coordinate diverse animal movements, both rhythmic (e.g. breathing, chewing, swimming, walking) and discrete (e.g. reflexes, reaching, sitting, jumping). Therefore, NCAP architectures that closely resemble neural circuit mechanisms could in principle be applied to a variety of additional movement types.

(3) *Circuit Modalities*. Biologically inspired ANN architecture may also provide useful inductive biases for upstream tasks involving perception and decision making. In animals, lower-level pattern generator circuits are modulated by higher-level control circuits. For instance, locomotion speed and direction are modulated by orientation and escape circuits in response to visual, auditory, and tactile stimuli. Such higher-level circuits could inspire NCAP architectures for additional modalities.

Ultimately, our work suggests a way of advancing AI and robotics research inspired by systems neuroscience and encourages future work in more complex embodied control.

Acknowledgments and Disclosure of Funding

Thanks to Sergey Shuvaev, Pavel Tolmachev, Liam McCarty, Polina Kirichenko, Pavel Izmailov, Daniel Barabasi, Christopher Langdon, and Josh Merel for insightful discussions and manuscript feedback. This work was supported by Schmidt Futures (N.X.B.); the G. Harold and Leila Y. Mathers Charitable Foundation (A.M.Z.); the Eleanor Schwartz Charitable Foundation (A.M.Z.); NIH grants S10OD028632-01 and RF1DA055666 (T.A.E.); and the Alfred P. Sloan Foundation Research Fellowship (T.A.E.).

References

J. Ausborn, N. A. Shevtsova, and S. M. Danner. Computational modeling of spinal locomotor circuitry in the age of molecular genetics. *International Journal of Molecular Sciences*, 22(13):6835, June 2021.

X. Bin Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020.

Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, 12:35, July 2018.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Kohd, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models. *arXiv:2108.07258 [cs]*, Aug. 2021.

J. H. Boyle, S. Berri, and N. Cohen. Gait modulation in C. elegans: An integrated neuromechanical model. *Frontiers in Computational Neuroscience*, 6, 2012.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

D. Bucher, G. Haspel, J. Golowasch, and F. Nadim. Central pattern generators. In *eLS*. John Wiley & Sons, Ltd, Nov. 2015.

T. Buschmann, A. Ewald, A. von Twickel, and A. Büschges. Controlling legs for locomotion—insights from robotics and neurobiology. *Bioinspiration & Biomimetics*, 10(4):041001, June 2015.

P. Cisek. Resynthesizing behavior through phylogenetic refinement. *Attention, Perception, & Psychophysics*, 81 (7):2265–2287, Oct. 2019.

S. M. Danner, N. A. Shevtsova, A. Frigon, and I. A. Rybak. Computational modeling of spinal circuits controlling limb coordination and gaits in quadrupeds. *eLife*, 6:e31050, Nov. 2017.

A. Demin and E. Vityaev. Learning in a virtual model of the C. elegans nematode for locomotion and chemotaxis. *Biologically Inspired Cognitive Architectures*, 7:9–14, Jan. 2014.

J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding sparse, trainable neural networks. *arXiv:1803.03635 [cs]*, Mar. 2019.

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

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

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

R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu. A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4082–4093. PMLR, July 2020.

D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, July 2017.

N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *arXiv:1610.05182 [cs]*, Oct. 2016.

N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver. Emergence of locomotion behaviours in rich environments. *arXiv:1707.02286 [cs]*, July 2017.

A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653, May 2008.

A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke. Policies modulating trajectory generators. *arXiv:1910.02812 [cs]*, Oct. 2019.

E. J. Izquierdo and R. D. Beer. An integrated neuromechanical model of steering in C. elegans. In *ECAL 2015: The 13th European Conference on Artificial Life*, pages 199–206, July 2015.

G. 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, Feb. 1987.

O. Kroemer, S. Niekum, and G. Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv:1907.03146 [cs]*, Nov. 2020.

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

M. Lechner, R. Hasani, M. Zimmer, T. A. Henzinger, and R. Grosu. Designing Worm-inspired Neural Networks for Interpretable Robotic Control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 87–94, Montreal, QC, Canada, May 2019. IEEE.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *arXiv:1504.00702 [cs]*, Apr. 2016.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019.

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

X. Liu, A. Rosendo, S. Ikemoto, M. Shimizu, and K. Hosoda. Robotic investigation on effect of stretch reflex and crossed inhibitory response on bipedal hopping. *Journal of The Royal Society Interface*, 15(140):20180024, Mar. 2018.

L. Luo. Architectures of neuronal circuits. *Science*, 373(6559), Sept. 2021.

J. Merel, D. Aldarondo, J. Marshall, Y. Tassa, G. Wayne, and B. Ölveczky. Deep neuroethology of a virtual rodent. *arXiv:1911.09451 [q-bio]*, Nov. 2019a.

J. Merel, M. Botvinick, and G. Wayne. Hierarchical motor control in mammals and machines. *Nature Communications*, 10(1):5489, Dec. 2019b.

J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess. Neural probabilistic motor primitives for humanoid control. *arXiv:1811.11711 [cs]*, Jan. 2019c.

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

P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, Kobe, May 2009. IEEE.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. *arXiv:1912.01703 [cs, stat]*, Dec. 2019.

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

I. A. Rybak, K. J. Dougherty, and N. 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, Sept. 2015.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864 [cs, stat]*, Sept. 2017.

G. P. Sarma, C. W. Lee, T. Portegys, V. Ghayoomie, T. Jacobs, B. Alicea, M. Cantarelli, M. Currie, R. C. Gerkin, S. Gingell, P. Gleeson, R. Gordon, R. M. Hasani, G. Idili, S. Khayrulin, D. Lung, A. Palyanov, M. Watts, and S. 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, Oct. 2018.

S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. In H. Kimura, K. Tsuchiya, A. Ishiguro, and H. Witte, editors, *Adaptive Motion of Animals and Machines*, pages 261–280. Springer-Verlag, Tokyo, 2006.

S. Schaal and N. Schweighofer. Computational motor control in humans and robots. *Current Opinion in Neurobiology*, 15(6):675–682, Dec. 2005.

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

Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, P. Trochim, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. Dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, Nov. 2020.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Vilamoura-Algarve, Portugal, Oct. 2012. IEEE.

J. J. Torres and P. Varona. Modeling biological neural networks. In G. Rozenberg, T. Bäck, and J. N. Kok, editors, *Handbook of Natural Computing*, pages 533–564. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Q. Wen, S. Gao, and M. 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, Oct. 2018.

J. Yu, M. Tan, J. Chen, and J. Zhang. A Survey on CPG-Inspired Control Models and System Implementation. *IEEE Transactions on Neural Networks and Learning Systems*, 25(3):441–456, Mar. 2014.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 2, Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Supplemental Material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figure 4,Figure 5,Figure 7.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Appendix A.
   (b) Did you mention the license of the assets? [Yes] See Appendix A.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code under MIT license.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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 Learning Algorithms

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 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.

A.3 Implementation

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

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

A.4 Hyperparameters

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

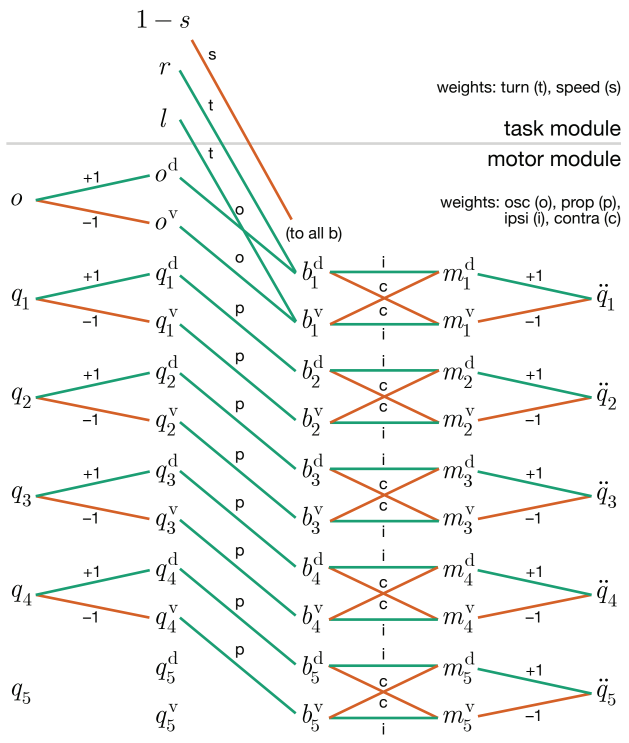
ES Algorithm Population size, 256; noise standard deviation σ, 0.02; L2 weight decay, 0.005; optimized, Adam; learning rate, 0.01; timesteps, 5e7.

NCAP Swimmer Oscillator, square wave, period 60 timesteps, width 30 timesteps.

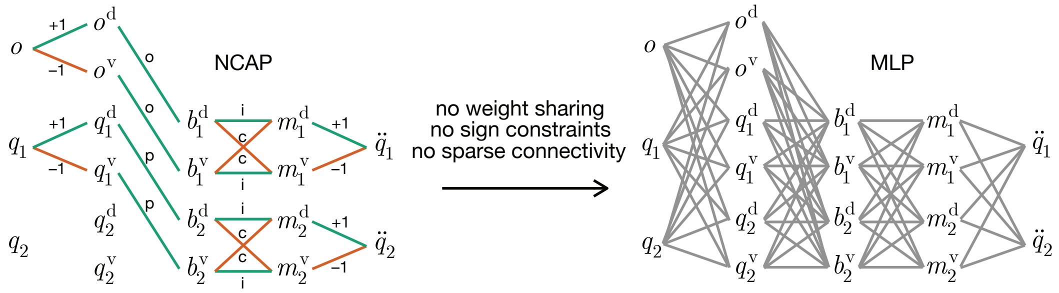
B Swimmer Architecture Details

Our NCAP architecture has the special property that it can be completely embedded within a fully connected MLP of 3 hidden layers and ReLU nonlinearities. This enables us to "interpolate" between our NCAP architecture and the MLP architecture, conducting a rigorous analysis of how various features of our architectural prior contribute to performance and learning.

By rearranging terms in the Swimmer network architecture diagram (Figure 3B), we arrive at the following network ($N = 5$ shown):



By removing weight sharing, sign constraints, and sparse connectivity, we arrive at a fully connected MLP of 3 hidden layers ($N = 2$ shown):



For $N = 5$, the resulting MLP has hidden layers of dimensions (12, 10, 10).