eLife

Reviewed Preprint
v1 • August 27, 2026
Not revised

For correspondence:
bm9751@princeton.edu
apv2@princeton.edu

Competing interests: This work was not supported by any dedicated funding. Princeton University has a patent pending for technology related to this manuscript.

Reviewing editor: Richard Naud, University of Ottawa, Canada

© 2026, Midler & Vazquez. This article is distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use and redistribution provided that the original author and source are credited.

Evolution imposes an inductive bias that alters and accelerates learning dynamics

Benjamin Midler, **Alejandro Pan Vazquez**
Princeton Neuroscience Institute, Princeton, United States

eLife Assessment

This article provides a computational study to understand how evolutionary selection could introduce structural priors into neural networks to improve learning. This **useful** investigation provides **incomplete** evidence that such joint learning is possible and has some counterintuitive features. However, the extent to which the models studied are distinct from those studied in the literature previously remains unclear.

https://doi.org/10.7554/eLife.111582.1.sa2

Abstract

The learning dynamics of biological brains and artificial neural networks are of interest to both neuroscience and machine learning. A key difference between them is that neural networks are often trained from a randomly initialized state whereas each brain is the product of generations of evolutionary optimization, yielding innate structures that enable few-shot learning and inbuilt reflexes. Artificial neural networks, by contrast, require non-ethological quantities of training data to attain comparable performance. To investigate the effect of evolutionary optimization on the learning dynamics of neural networks, we combined algorithms simulating natural selection and online learning to produce a method for evolutionarily conditioning artificial neural networks, and applied it to both reinforcement and supervised learning contexts. We found the evolutionary conditioning algorithm, by itself, performs comparably to an unoptimized baseline. However, evolutionarily conditioned networks show signs of unique and latent learning dynamics, and can be rapidly fine-tuned to optimal performance. These results suggest evolution constitutes an inductive bias that tunes neural systems to enable rapid learning.

1 Introduction

Both neuroscience and machine learning focus on neural systems—biological brains or artificial neural networks—that learn using online mechanisms like synaptic plasticity or backpropagationbased gradient descent [24, 4]. However, neural networks are often trained from a random initial state with no preexisting knowledge, whereas biological brains, through natural selection, have undergone an additional form of optimization that may produce innate neural structures (Fig. 1).

The objective function of natural selection is to maximize fitness, or the ability to survive and reproduce [6]. A cornerstone of the brain's contribution to fitness is its ability to engage in online learning: a brain that can adapt to its environment will likely outlast one that cannot. As such, there is evolutionary pressure in favor of developing a more plastic neural system that can rapidly adapt to its environment. The effect of online learning on the speed and course of evolution is known as the Baldwin effect [2, 22]. In computational terms, it can be thought of as reciprocal optimization loops: evolution optimizes the initial state of the network to maximize learning potential, and online learning, in turn, factors into the fitness objective function that guides future evolutionary development.

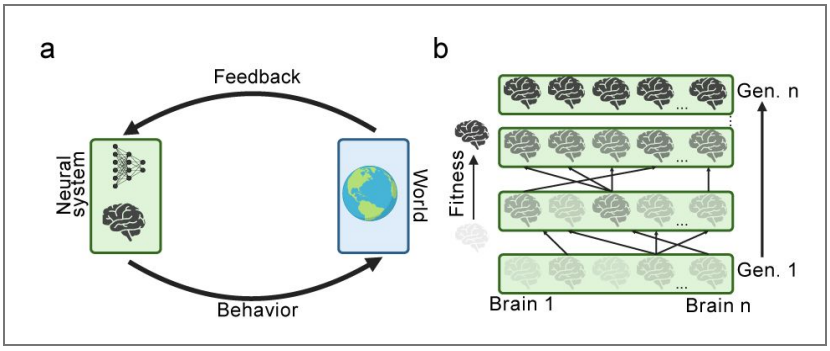

Figure 1. Brains are optimized by natural selection and online learning.

(A) Brains and artificial neural networks engage in online learning by producing behaviors and receiving environmental feedback to guide learning. (B) The fitness of biological brains is optimized over evolutionary timescales by natural selection. This constitutes an additional form of optimization on the initial state of a brain, endowing it with inbuilt structures for innate reflexes and enhanced learning potential. In biology, evolutionary optimization and online learning co-exist as evolutionary pressures act upon the genetic information that produces a new brain that then engages in online learning.

There is an important contrast between formulating evolution's impact on learning as optimizing the initial condition of a network and how it is often used in deep learning, namely through genetic algorithms [14, 21, 28, 37]. These algorithms, which task-optimize neural networks by evaluating a population of networks on an objective function and seeding the next generation by applying stochastic noise to the weights of the best performers, are able to learn complex tasks comparably to backpropagation-based gradient descent [30, 25]. In this application of evolutionary logic to machine learning, natural selection is the sole optimization mechanism and is thus analogous to freezing synaptic plasticity at birth. This is in stark contrast to the biological influence evolution has on the brain, as it acts upon the heritable information used to develop a new organism that may then engage in online learning before passing-on heritable information to the next generation. Crucially, online learning has no direct impact on the heritable information transferred between generations; its influence is strictly limited to the fitness objective function that determines the quantity of offspring produced by an individual.

It is worth noting that there are many examples of genetically encoded innate neural computations that manifest through, for example, neuronal connectivity patterns, dendritic morphology, and firing dynamics [20]. The drosophila brain, for instance, is known to be highly stereotyped, with much of its connectivity driven by transcription factor expression [40, 38]. Additionally, even for sophisticated mammalian behaviors, the line between innateness and learning may be blurred, as many social behaviors have both innate and learned components [34]. As such, we do not claim that evolution cannot nor does not optimize neural systems to perform complex operations—this is clearly false—merely that there is an interplay between these two learning mechanisms that yields unique dynamics.

There have been various efforts to more accurately model brain evolution by, for instance, modeling sequential steps of genetic and gradient descent-based optimization [36], introducing a compression step to simulate a genetic bottleneck [27], and by iteratively setting and freezing binary connections [13]. Each of these efforts demonstrated improvements in optimization speed, consistent with the Baldwin effect; however, they involve the direct transfer of learned weights from one generation to the next. This is a Lamarckian formulation in which acquired traits are transferred between generations. While recent advances in genetics have not ruled Lamarckian evolution impossible [18], it is not a primary mechanism of evolutionary optimization. The relationship between evolutionary optimization and online learning—and the impact this interplay has on a neural network—thus remains unclear.

To address this, we evolutionarily condition artificial neural networks by using a genetic algorithm and gradient descent in distinct steps. Our approach selectively applies evolutionary pressure to information passed between generations, and limits the role of online learning to parameterizing the objective function. This addresses the gap left by prior work as learned network weights are not transferred between generations. We found our model to yield unique latent learning dynamics and significant improvements in the speed of learning, thereby suggesting evolution imposes an inductive bias that promotes the rapid acquisition of new knowledge.

2 Results

2.1 Evolutionary conditioning implementation

To assess the effect of merging computational models of evolutionary optimization and gradient descent, we developed a new algorithmic process, dubbed evolutionary conditioning (EC). EC is architecturally agnostic and we applied it to networks in reinforcement and supervised learning contexts.

The core of the EC process is similar to that of existing genetic algorithms: we treat the weights of a deep neural network as a genotype which is inherited by subsequent generations [21, 37, 30]. The key departure from a standard genetic algorithm is that, rather than directly evaluating each network in the generation, we independently fine-tune network weights on a rollout of the task using backpropagation-based gradient descent. We then evaluate the fine-tuned networks to

determine fitness and probabilistically spawn the next generation by applying gaussian noise to the weights of the best performing parent networks. Crucially, we use the parent's weights prior to fine-tuning to form the child network. This dissociates the evolutionary and gradient-descent optimization loops, thereby constraining the influence online learning has on the evolutionary process to the fitness objective function. In effect, this causes the genetic algorithm to optimize the network's potential to learn the task rather than directly optimizing the network's ability to perform the task. While these two objectives can converge—it is easy to learn something if you already possess the knowledge innately—we find this to not be the case.

2.2 Reinforcement learning reveals latent behavioral structure

We first assessed EC on a reinforcement learning task in which the network must learn to balance a pole by applying lateral force to the platform on which the pole stands [32, 3]. This is a simple and well-defined benchmark for motor control, which is a fundamental operation for any brain (Fig. 2a).

We trained a neural network to perform this task using both a standard genetic algorithm (GA) and backpropagation-based gradient descent (SGD)—thus decomposing EC into its constituent evolutionary and online learning processes. Both algorithms were able to train the network to perform at ceiling, defined as the pole remaining upright for 1,000 time steps (both models mean=1,000, SD +/- 0 over 50 episodes; Fig. 2b). We then trained the network using EC for 2,000 generations—equal to the number used for the GA—and with a roll-out of 100 episodes for fine-tuning—5% of the total used for SGD. Whereas both SGD and the GA were able to consistently learn the task, EC consistently failed: its performance was indistinguishable from an untrained network (EC mean=5.28, SD +/-1.47; untrained mean=6.88, SD +/-4.06; Mann-Whitney U test Bonferroni corrected p=1.0 over 50 episodes; Fig. 2b). This prompted us to investigate the networks' behavior. We uniformly sampled 10,000 task observations—vectors detailing features like cart position and velocity—and recorded each network's behavioral responses to the observations. In other words: given a random task state, what does the network do? This demonstrated that, even though the EC and untrained networks performed indistinguishably, EC network behavior followed a flatter distribution far more similar to that of the GA network (EC vs untrained network Kolmogorov-Smirnov distance=0.4103; EC vs GA Kolmogorov-Smirnov distance=0.0426; Fig. 2c), and that the GA and SGD networks, despite both performing at ceiling, displayed significantly different behavioral distributions (Kolmogorov-Smirnov distance=0.1254, Bonferroni corrected p<4e-67).

To better understand if these distributional differences are reflected in underlying behavioral strategy, we again passed sampled task states to each network and recorded behavioral responses, but this time we did so simultaneously for two networks to assess their congruence. We abstracted over the magnitude of network output, thus binarizing behavior as a move to the left or the right. Behavioral congruence between networks was expressed as a matrix, with matrix cells giving the proportion of observation inputs that produced a given combination of left or right outputs from the networks (Fig. 2d). The matrix diagonal was then summed to give the proportional congruence between networks. We repeated this 100 times to form congruence distributions for each combination of networks, and produced empirical null distributions by shuffling the observation order for each network combination (Fig. 2e). As expected, SGD and GA networks had both high rates of internal congruence and were significantly congruent—albeit slightly less so—with each other (SGD internal congruence Welch's t-test Bonferroni corrected p<2e-166; GA internal congruence Welch's t-test Bonferroni corrected p<7e-156; SGD vs GA congruence Welch's t-test Bonferroni corrected p<3e-142). Of greater interest was the EC network having a significantly elevated rate of internal congruence and significantly elevated congruence with both SGD and GA networks (EC internal congruence Welch's t-test Bonferroni corrected p<2e-151; EC vs SGD Welch's t-test Bonferroni corrected p<4e-12; EC vs GA Welch's t-test Bonferroni corrected p<2e-32). These results show the EC network, despite performing the task indistinguishably from an untrained baseline, evidenced a consistent behavioral strategy that overlapped significantly with networks

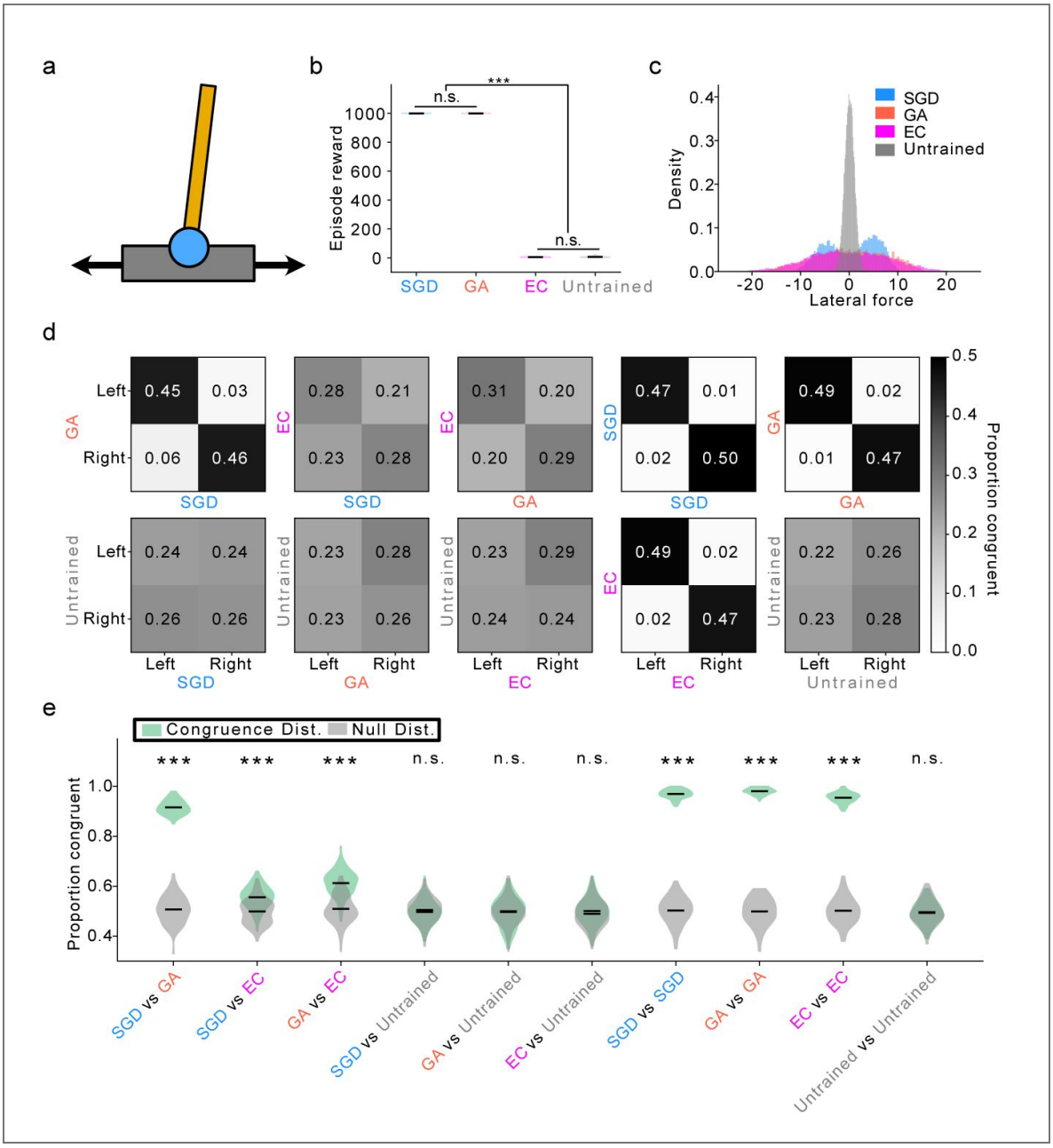

Figure 2. EC fails to task-optimize but shows signs of latent learning.

(A) Schematic of the reinforcement learning problem. A pole is balancing on a platform that can be moved laterally. Networks are trained to move the platform to keep the pole balancing upright. (B) Task performance per model. Each reward is a timestep during which the pole is balanced. Max per episode is 1,000, n=50 episodes (Mann-Whitney U test ***p<1e-18 Bonferroni corrected). (C) Distribution of network actions in response to 10,000 uniformly sampled task states (vectors of cart position, velocity, pole angle, and angular velocity). EC behavior is more similar to the GA even though performance is no different than an untrained network. Note: GA and EC distributions are highly overlapping. (D) Example behavioral congruence matrices between each model pair. (E) Congruence distributions (green) versus shuffled null (gray), n=100 matrices as in D. EC is internally consistent and significantly similar to successfully trained networks, indicating latent learning (Mann-Whitney U test ***p<4e-12 Bonferroni corrected).

fully optimized on the motor control task. This indicates that nesting dissociated loops of evolutionary optimization and online learning yield a unique optimization problem with signs of latent learning.

2.3 Evolutionary conditioning dynamics are distinct from learning dynamics

Reinforcement learning is a fundamental paradigm in neuroscience and machine learning as it is central to how animals and artificial neural networks learn behavioral responses to environmental challenges [31]. Interpreting network dynamics during reinforcement learning, however, is difficult [8]. Therefore, in order to clarify how EC network dynamics may be distinct from those of SGD and GA networks, we applied all three to a sequence-to-sequence supervised learning problem in which a network learns associations between a set of items and their attributes (Fig. 3a). This is a semantic cognition task with several properties useful for forming mechanistic hypotheses for how a neural network acquires information [26, 19]. Specifically, we applied singular value decomposition to the items-to-attributes dataset, yielding a set of modes, the weightings of each item onto the modes, and the singular values that correspond to mode precedence (Fig. 3b-d). These modes correspond to the hierarchical similarity structure between the items in the dataset, arranging them into superordinate (e.g. plants and animals) and subordinate (e.g. birds and fish) categories (Fig. 3e). Networks trained on this semantic cognition task acquire information sequentially, following mode order: networks first learn to differentiate the highest-order categories before progressing to lower-orders and finally differentiating the items themselves [26, 19].

We trained networks on the semantic cognition task using SGD and the GA, confirming that the algorithms could task-optimize (Fig. 3f-g). Learning dynamics were assessed by taking the vectors of hidden-layer activity evoked by categorical inputs and computing the euclidean distance between them. When a level of the categorical hierarchy is learned, the euclidean distance between the category vectors increases. Using this method, we confirmed that the learning dynamics of networks trained with SGD and the GA followed the sequential order given by the dataset's similarity structure and singular value decomposition (Fig. 3i-j). We then trained the network using EC. As with the reinforcement learning problem, the EC-trained network did not task-optimize: over the course of 4,000 generations its loss remained flat or increased (Fig. 3h). Moreover, the EC network's learning dynamics were markedly different from those of the GA and SGD networks. Whereas those algorithms yielded the expected sequence of learning higher-order categories before lower-orders, EC concurrently differentiated all levels of the hierarchy (Fig. 3k). This indicates EC not only produces latent learning dynamics, which was also indicated by the reinforcement learning task, but that those latent dynamics are distinct from other learning algorithms and do not respond to the same mathematical regularities in the training data.

2.4 Evolutionary conditioning facilitates rapid learning

Thus far we have found that EC displays latent learning dynamics in reinforcement and supervised learning tasks. A core prediction of the Baldwin effect is that the evolutionary process yields more rapid online learning [7, 13]. To test this prediction, we took EC networks spaced 250 generations apart and used backpropagation-based gradient descent to fine-tune them on the semantic cognition task (Fig. 4a). This is analogous to an organism being born the product of generations of evolutionary optimization and then engaging in online learning. We found that the number of fine-tuning epochs required for EC networks to reach criterion decreased monotonically from 1,597 (SD +/-159) for a network with no EC—only fine-tuning—to 148 (SD +/-0) for a network with 4,000 generations of EC (Mann-Whitney U test $p<5e-38$; Fig. 4b-e). This order of magnitude increase in training speed constitutes computational support for the Baldwin effect, and indicates evolutionary optimization can accelerate online learning even when no gradient information nor gradient-optimized weights are transferred between generations.

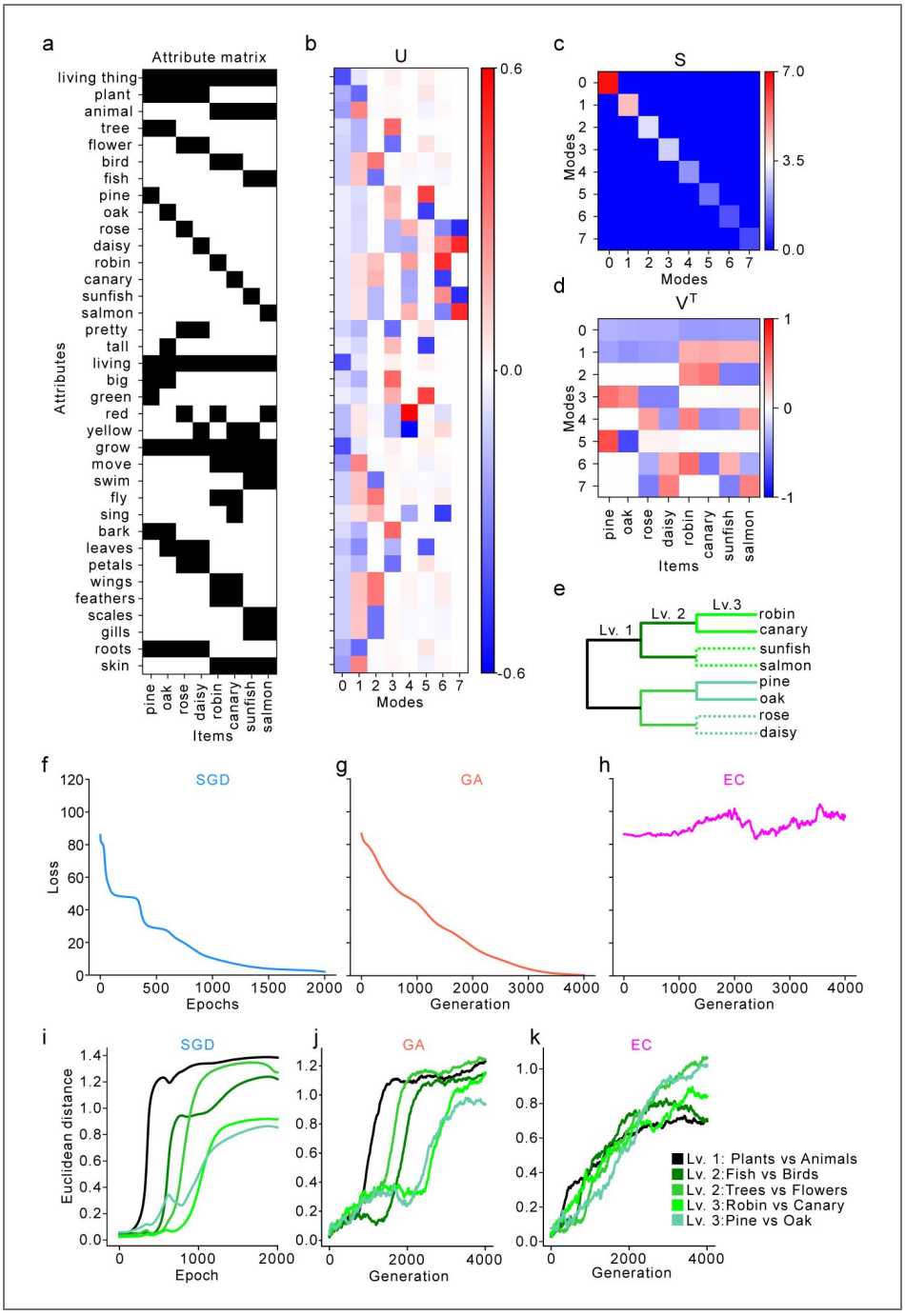

Figure 3. EC representational dynamics are unique and do not respond to known regularities in training data.

(A) The items-to-attributes dataset used in the semantic cognition dataset. (B) The U matrix produced by the singular value decomposition of the dataset. The U matrix shows the loadings of each attribute per mode. (C) The S matrix produced by the singular value decomposition of the dataset. The values of the diagonal, or singular values, correspond to the precedence of the modes. (D) The VT matrix produced by the singular value decomposition of the dataset. It shows the loadings of each item onto the modes, illustrating how the modes correspond to categorical differentiations (e.g. mode 1 splits plants and animals, mode 2 splits birds and fish). (E) Schematic of dataset similarity structure divided into categorical levels. Solid branches are those shown in I-K, dotted branches are not shown but are equivalent to those from the same hierarchical level. (F-H) Loss curves of each model during training. Both SGD and GA task-optimize, but not EC. (I-K) Representational dynamics of each model throughout training. Both SGD and GA show sequential learning dynamics in which superordinate categories are learned before subordinate. EC, however, learns to distinguish all categorical levels concurrently.

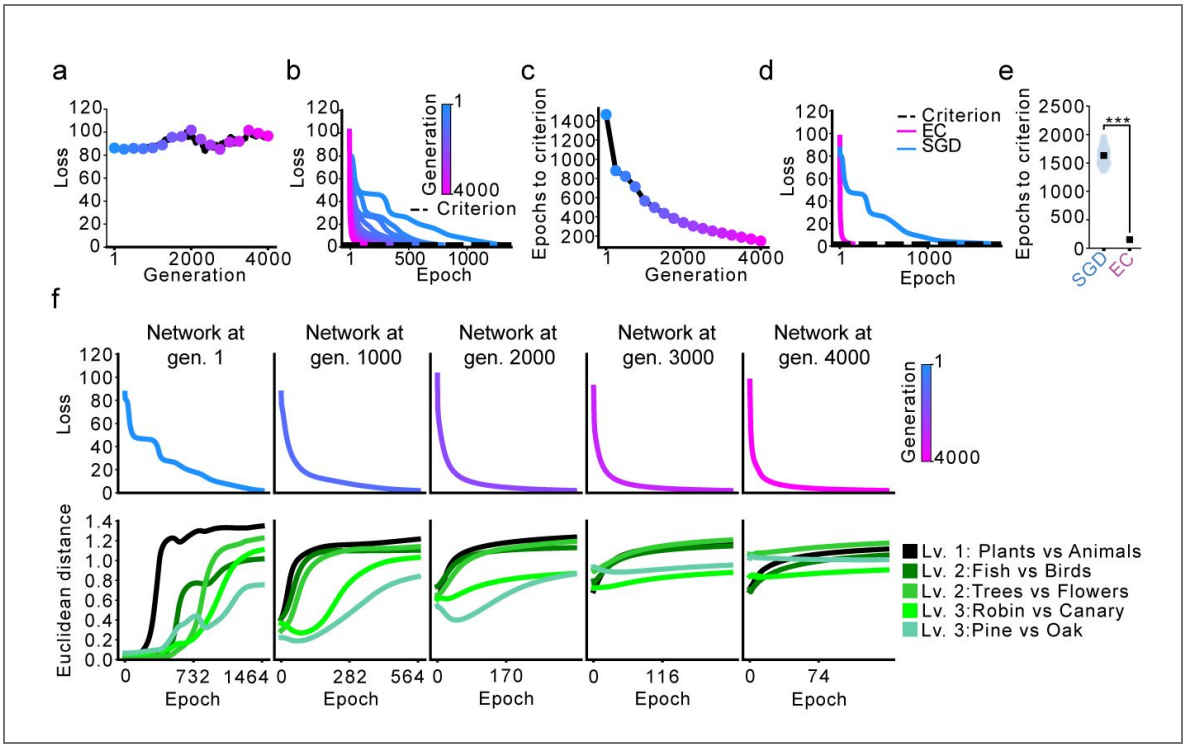

Figure 4. EC speeds fine-tuning and alters gradient descent representational dynamics.

(A) EC loss curve with colored dots indicating network snapshots taken form throughout training. (B) Loss curves of network snapshots from A being trained to criterion. (C) Same as in B but each network snapshot is expressed as epochs of training required to reach criterion. Even though loss during EC is flat or elevated (shown in A), it accelerates fine-tuning by an order of magnitude. (D) Loss curves for SGD and EC networks trained to criterion. (E) Comparing epochs to train to criterion for a network trained via SGD and a network pre-trained with EC then fine-tuned (Mann-Whitney U test Bonferroni corrected ***p<5e-38). (F) Loss curves (top) and representational dynamics (bottom) for network snapshots from throughout EC as they were fine-tuned. Dynamics are unique not only during EC, but during fine-tuning as well.

2.5 Evolutionary conditioning alters gradient descent learning dynamics

Though the EC process itself displays unique representational dynamics distinct from those of networks trained with SGD or the GA, we were curious how EC, as a form of pre-training, altered the learning dynamics of gradient descent during fine-tuning. Does EC push the network to a parameter subspace that, when fine-tuned, shows the same learning dynamics as the SGD or GA networks but at a faster timescale, or do the dynamics remain unique? Thus, as a final test, we compared the learning dynamics of networks from throughout the EC process as they were fine-tuned using gradient descent (Fig. 4f). Doing so revealed that, early in EC, learning dynamics followed a sequential pattern similar to the SGD and GA networks (Fig. 4f left column). As EC progressed, however, the pattern changed. At intermediate generations the top two levels of the hierarchy were collapsed into a single learning phase (Fig. 4f middle columns), and, at late generations, representational distances for all categorical levels remained stable and intermixed (Fig. 4f right column). This is despite the fact that, as loss remains elevated throughout EC, the networks saw similar levels of performance improvement during fine-tuning. These simulation results further support the notion that evolution introduces latent representational dynamics into the weights of neural networks, and offers a potential explanation for why brains optimized by generations of evolution can display distinct and more rapid learning dynamics than computational models.

3 Discussion

Evolutionary models have yielded useful insights into the developmental course of neurobiology and cognition [21, 28]. For instance, genetic algorithms have been used to form de novo hypotheses on the emergence of the command neurons found in *aplysia*, lobsters, and crayfish, demonstrating that computational models of evolution can contain biological insight [21, 1]. As a complementary point, a recent study ported a connectome of *C. elegans* into a simulated worm, finding that the biological synaptic weights constitute a high-performing and data-efficient baseline for training the model worm to swim, demonstrating biology can be a useful prior in machine learning [5].

This latter example is indicative of the *tabula rasa* problem: that artificial neural networks—which purport to be at least superficially based on the brain—are often randomly initialized as a blank slate [39]. This is in contrast to the generations of evolutionary optimization that endow biological brains with an innate structure upon which online learning builds. We used a novel model of neural development that dissociates evolutionary optimization from online learning to investigate the evolution-driven emergence of innate structure in artificial neural networks. We demonstrate that evolution conditions a neural network to display latent behavioral and computational dynamics that, while undetectable in baseline performance, constitute an inductive bias that significantly alters learning dynamics, violates theories of how networks respond to mathematical regularities in training data, and speeds learning by an order of magnitude. Evolutionary conditioning may therefore help explain how biological brains are able to learn new information from only a few examples whereas computational models are notoriously data inefficient [15, 11, 9].

While we apply this model to both reinforcement and supervised learning tasks, our focus was on using small, relatively tractable problems that would provide mechanistic insight into computational and behavioral dynamics. Going forward, it will be useful to investigate this formulation of neural evolution by applying it to a broad range of computational problems and models, to use evolutionary algorithms that jointly act on network weights and architectures [29], and to concurrently evolve the physiology of an agent and its neural network controller [10]. Doing so will help clarify the level of neural structure upon which evolutionary pressures operate. In our model, online learning and evolutionary optimization were applied to the same weights—albeit in distinct optimization steps. While network nodes in the semantic cognition simulation ought to be thought of as coarse connections between abstract cognitive modules rather than

synapses between neurons—an important distinction as synaptic weights are not genetically encoded but coarse axon wiring can be [16, 12]—it will be important to understand how evolutionary optimization and online learning can cooperate across multiple levels of brain structure to produce optimal solutions.

In sum, we propose a novel algorithm that models evolution and online learning as distinct steps. We demonstrate how this algorithm conditions artificial neural networks to display latent behavioral and computational dynamics that do not follow known patterns of network learning, yet accelerates task-optimization by an order of magnitude. These results suggest evolution imposes an inductive bias on neural networks that significantly alters and accelerates learning dynamics.

A Technical Appendices and Supplementary Material

A.1 Reinforcement learning: task

The reinforcement learning task was the MuJoCo implementation of the cart pole motor control problem [33, 3]. We used default environment parameters (uniform sampling of initial state, max episode length=1,000 time steps, reset noise scale=0.01). The task objective is to keep a pole balanced upright by applying lateral force to the platform on which the pole is balancing. A successful time step is defined as $|\text{angle}|<0.2$ radians, and returns a reward of +1. The episode terminates if angle exceeds 0.2 radians, or the maximum episode length is reached, whichever occurs first. The task produces four observation values each time step: cart position along the linear slider (meters), cart velocity (meters/second), pole angle (radians), and pole angular velocity (radians/second). All observation values are continuous and range from negative to positive infinity. The task action space has one value: lateral force applied to the cart (newtons). Force values are continuous between -3 (maximum leftward force) and +3 (maximum rightward force).

A.2 Reinforcement learning: gradient descent implementation

The stochastic gradient descent (SGD) model trained on the cart pole reinforcement learning problem was a deep neural network that takes task observations as an input and produces as an output the mean () and standard deviation () of a distribution that is sampled to produce the model action. Loss is computed by the REINFORCE algorithm which is then used to update network weights via backpropagation [35, 24].

The model was implemented using PyTorch [23], and is structured as such:

$$\begin{gathered}\text{Input}: \ s \in \mathbb{R}^{\text{obs\_dim}} \\ \text{Hidden Layer 1}: \ h_1 = \ \text{ReLU}\left(W_1 s + b_1\right) \\ \text{Hidden Layer 2}: \ h_2 = \ \text{ReLU}\left(W_2 h_1 + b_2\right) \\ \text{Output Layer (Mean)}: \ \mu = W_\mu h_2 + b_\mu \\ \text{Output Layer (Log SD)}: \ \log \sigma = \text{Learnable Parameter} \\ \text{Standard Deviation: } \sigma = \exp\left(\log \sigma\right)\end{gathered}$$

The process for producing a model action, *a*, given an input state vector, *s*, and using the neural network as the policy is thus:

$$\begin{gathered}\mu, \sigma = \text{policy}\left(s\right) \\ \text{Distribution}: \mathcal{N}\left(\mu, \sigma\right) \\ \text{Sampled Action}: a \sim \mathcal{N}\left(\mu, \sigma\right) \\ \text{Log Probability}: \ \log p\left(a\right) = \sum_i \log \mathcal{N}\left(\mu_i, \sigma_i\right)\end{gathered}$$

Note that the log probability of the selected action is computed for subsequent use in the loss function.

The weights of the network are trained via backpropagation [24]. To produce the loss via the REINFORCE algorithm, the log probabilities of rewards are summed with return values, which are the discounted rewards received after an action. Specifically:

$$\text{Returns}: \ R_t = \sum_{k=0}^{T-t-1} \gamma^i r_{t+i}$$
$$\text{Normalized Returns}: \ \widehat{R} = \frac{R - mean(R)}{SD(R) + \epsilon}$$
$$\text{Loss}: \ L = -\sum_t \log p\left(a_t\right) \widehat{R}_t$$

Here, R_t is the return value at time step t, T is the number of total time steps in the episode, ϵ is a small value to prevent division by zero, and r_{t+i} is reward at time step $t+i$. Models were trained with the discount factor γ = 0.99 and a learning rate of $1e$ - 3 using the Adam optimizer [17].

A.3 Reinforcement learning: genetic algorithm implementation

The genetic algorithm (GA) model was implemented using the same network architecture as the SGD model: the deep neural network takes task observations as inputs and produces the mean and standard deviation of a distribution that is sampled to produce the action output. The only difference between the models was the training mechanism.

Specifically, for the GA model, a population of networks was initialized with random weights, constituting the first generation. Each model in the generation was then evaluated on an episode of the cart pole task, with network fitness quantified as the reward received in the episode.

The reward of each network (policy), π, is given by the sum of rewards received on each time step, t, of the episode, with T being the number of time steps in the episode.

$$\text{Reward of policy } \pi_i: \ R\left(\pi_i\right) = \sum_{t=0}^{T} r_t$$

To spawn the next generation of networks (child networks), the best performing parent networks are sampled probabilistically—and with replacement—according to their task performance. A selected parent model has gaussian noise applied to its weights to produce a child model.

$$\text{Reward of Parent Model } \pi_i: R\left(\pi_i\right)$$
$$\text{Weight of parent Model } \pi_i: w_i = R\left(\pi_i\right) - R_{\min} + \epsilon$$
$$\text{Probability of Selecting Parent Model } \pi_i: p_i = \frac{w_i}{\sum_{j=1}^{k} w_j}$$

Here, $R(\pi_i)$ is the reward for parent model π_i, R_{min} is the minimum reward received by the k = 3 top performing parent models, wi is the probabilistic weight of selecting parent model π_i, ϵ is a small constant used to ensure non-zero probability of selecting a parent model for seeding a child model, and p_i is the probability of selecting parent model π_i for seeding a child model.

The process of seeding a child network, π'_i, from parent network π_i consists of adding gaussian noise to the weights of the parent model, with *mean* = 0 and *standarddeviation* = 0.01.

$$\pi'_i = \pi_i + \mathcal{N}\left(0, 0.01\right)$$

An additional important detail is that we employed generational annealing to create a dynamic schedule of population sizes across generations. This enables large populations at early generations when robustly sampling the space of possible solutions is important, and small populations at later generations for computational efficiency. Specifically, the number of networks at a given generation g, $P(g)$, is given by

$$\text{Population Size at Generation } g : P(g) = P_{start} \cdot \exp\left(\left(\frac{g}{G-1}\right)^{\lambda} \cdot \log\left(\frac{P_{end}}{P_{start}}\right)\right)$$

Here, P_{start} and P_{end} set the initial and final population sizes and λ is a decay term used to set the annealing schedule. It was set to 0.1 whereas P_{start} and P_{end} were 1,000 and 10, respectively. G is the number of generations.

A.4 Reinforcement learning: evolutionary conditioning implementation

The evolutionary conditioning (EC) implementation combines those of the SGD and GA models. Put simply, the EC model uses the GA selection mechanism, but, after seeding a new generation, uses the SGD REINFORCE implementation to fine-tune network weights on a rollout of the cart pole task. Networks with fine-tuned weights are then evaluated, as in the GA implementation, and evaluation scores are used to sample networks for seeding the next generation. However, the gaussian noise is applied to network parameters prior to fine-tuning with gradient descent.

A.5 Reinforcement learning: analyses and statistics

A.5.1 Model performance comparison

Model performance was compared by initializing 50 random instances of the cart pole task and recording the rewards received by models on the 50 episodes. The same episode initializations were passed to each model in the comparison. Reward per episode is defined as the number of time steps during which $|angle| < 0.2$ radians, with a maximum of 1,000 time steps. Two-sided Mann-Whitney U tests were used for pair-wise comparisons between model performances, and p values were Bonferroni corrected for multiple hypotheses by multiplying p values by the number of tests being performed.

A.5.2 Sampling observations

To sample observations for assessing model behavior, 10,000 random numbers were sampled for each of the cart pole task's four observational variables (cart position, cart velocity, pole angle, and angular velocity). The task technically allows for observational variables to be infinite. To make sampling tractable, we used a range of −1 to 1, which was empirically observed to capture the range of task observations produced by trained and untrained models alike.

A.5.3 Action distributions

To form action distributions produced by the sampled observations, the same observations were passed to each model, and the output action was recorded. Density distributions were plotted as histograms with 75 bins and compared using two-sample Kolmogorov-Smirnov tests. Bonferroni correction was used to compensate for multiple tests.

A.5.4 Congruency matrices

To form action congruency matrices between models, 1,000 observations were sampled as before and passed to two models in concert. Model actions in response to the given observation were recorded as leftward (action < 0) or rightward (action > 0). The cells of the matrix are populated by taking the number of observations that produced each combination of leftward and rightward outputs from the models (e.g. if an observation yielded a rightward move from one model and a leftward move from the other) and were normalized by the number of observations to produce a proportion.

A.5.5 Congruency distributions

To form congruency distributions, the same process as above was repeated 100 times, with each run containing 100 observations. The matrix diagonal for each model pair on each run was then summed to produce the proportion of observations that yielded congruent actions from both models. The distributions are formed of these congruency values over the 100 runs. To make null

distributions for statistical comparisons, the process was repeated but with the observations passed to one of the two models randomly shuffled. Statistical significance was established using Welch's t-test, and p values were corrected for multiple hypotheses testing using Bonferroni correction.

A.6 Semantic cognition: task

The semantic cognition task has been previously described in detail [26, 19]. In short, it requires a network to learn associations between items and their attributes, joined by a series of prepositions. The dataset is thus composed of binary input vectors for a set of items and a preposition that links the item to some of its attributes. The preposition input constrains the item-attribute association. For instance, if the input item is "canary" and the input preposition is "can", the target output attributes are "grow", "move", "fly", and "sing". The "canary" item has other attributes, such as having wings, but because "can" is the wrong preposition to link "canary" and "wings", the "wings" output unit is trained to not activate when "canary" and "can" are the inputs ("wings" is an active attribute when "canary" is paired with the preposition "has"). When comparing hidden layer representations between items, comparisons are averaged across all prepositions for the given item.

A.7 Semantic cognition: stochastic gradient descent implementation

The SGD model is a deep neural network that takes items and prepositions as inputs and is trained to output a binary vector for the associated attributes. The model was implemented in PyTorch and used sigmoid nonlinearities, the stochastic gradient descent optimizer, and mean-squared error [23]. Weights were initialized between −0.1 and +0.1 and the model was trained with backpropagation using a learning rate of 0.1 [24]. The model was trained for 2,000 epochs.

A.8 Semantic cognition: genetic algorithm implementation

The GA model used the same architecture, initialization procedure, nonlinearities, and task formulation as the SGD model, only it was trained using the genetic algorithm instead of backpropagationbased gradient descent. For the GA, populations of 100 networks were initialized, evaluated on the semantic condition dataset, with task loss used to represent network fitness that sets sampling probabilities for seeding child networks. This process is the same as it was for the reinforcement learning problem. As with the reinforcement learning formulation, the top k models used to seed the next generation was set to 3 and the standard deviation for the gaussian noise applied to networks was 0.01. The model was trained for 4,000 generations.

A.9 Semantic cognition: evolutionary conditioning implementation

As with the reinforcement learning implementation, the EC model combined the training methodologies of the SGD and GA models such that the GA, instead of using loss of the networks initialized as part of each generation, used the loss of the networks after being fine-tuned on the task with backpropagation-based gradient descent. In this case, networks were fine-tuned on the task for 100 epochs per generation for 4,000 generations. Crucially, the genetic algorithm hyper-parameters (e.g. k) were the same as for the GA, and the fine-tuning hyper-parameters (e.g. learning rate) were the same as for the SGD model, with the exception of the number of training epochs.

A.10 Semantic cognition: analyses and statistics

A.10.1 Learning dynamics

To assess learning dynamics, vectors of hidden layer activity were extracted for each item input to the model and euclidean distances were computed between vectors to quantify the model's ability to learn distinctions between input patterns. For categorical comparisons (e.g. plants vs animals),

distances were averaged across the constituents of the category. This process was repeated for model snapshots taken throughout training to trace the learning dynamics of the model.

A.10.2 Training to criterion

When assessing the time it takes to train a model to criterion, a network, or a snapshot taken from during training, was fine-tuned using backpropagation-based gradient descent until loss equals a set threshold. For all tests the threshold was set to 2. This value was chosen as it corresponds to accuracy >99% and SGD models were reliably able to attain it within 2,000 epochs of training and the GA model within 4,000 generations. Testing was performed with slightly different criterion values and the same trends were observed. Statistical significance was assessed using a two-sided Mann-Whitney U test. No correction for multiple hypothesis testing was applied as only one statistical comparison was performed per analysis.

A.11 Compute resources

All analyses and simulations were performed using a consumer-grade laptop (Apple MacBook Pro, 2023) with the M3 Max integrated CPU/GPU and with 48 GB of memory.

Data availability

No datasets were involved with this work.

Additional information

Author ORCID iDs

Benjamin Midler: https://orcid.org/0000-0001-9954-366X

References

[1] **Aharonov-Barki R,** Beker T, Ruppin E (2001) Emergence of memory-driven command neurons in evolved artificial agents. *Neural Comput* **13**:691-716 https://doi.org/10.1162/089976601300014529 | PubMed

[2] **Baldwin J** (1896) A new factor in evolution. *Am. Nat* **30**:441-451 https://doi.org/10.1086/276408

[3] **Barto Andrew G,** Sutton Richard S, Anderson Charles W (1983) Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern* **SMC-13**:834-846 https://doi.org/10.1109/tsmc.1983.6313077

[4] **Benfenati Fabio** (2007) Synaptic plasticity and the neurobiology of learning and memory. *Acta Biomed* **78**:58-66 PubMed

[5] **Bhattasali Nikhil X,** Zador Anthony M, Engel Tatiana A (2022) Neural circuit architectural priors for embodied control. *arXiv* https://doi.org/10.48550/arxiv.2201.05242

[6] **Darwin Charles,** Kebler Leonard (1859) *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life* London.

[7] **Dennett Daniel C** (1993) *Consciousness Explained* England: Penguin Books, Harlow.

[8] **Dethise Arnaud,** Canini Marco, Kandula Srikanth (2019) Cracking open the black box: What observations can tell us about reinforcement learning agents. In: Proceedings of the 2019 Workshop on Network Meets AI & ML - NetAI'19. New York, USA: ACM Press.

[9] **Graves Alex,** Wayne Greg, Reynolds Malcolm, Harley Tim, Danihelka Ivo, Barwińska Agnieszka Grabska-, Colmenarejo Sergio Gómez, Grefenstette Edward, Ramalho Tiago, Agapiou John, *et al.* (2016) Hybrid computing using a neural network with dynamic external memory. *Nature* **538**:471-476 https://doi.org/10.1038/nature20101 | PubMed

[10] **Gupta Agrim,** Savarese Silvio, Ganguli Surya, Fei-Fei Li (2021) Embodied intelligence via learning and evolution. *Nat. Commun* **12**:5721 https://doi.org/10.1038/s41467-021-25874-z | PubMed

[11] **Hafner Danijar,** Lillicrap Timothy, Ba Jimmy, Norouzi Mohammad (2019) Dream to control: Learning behaviors by latent imagination. *arXiv* https://doi.org/10.48550/arxiv.1912.01603

[12] **Hassan Bassem A,** Hiesinger P Robin (2015) Beyond molecular codes: Simple rules to wire complex brains. *Cell* **163**:285-291 https://doi.org/10.1016/j.cell.2015.09.031 | PubMed

[13] **Hinton Geoffrey E,** Nowlan Steven J (1987) How learning can guide evolution. *Complex Systems* **1**

[14] **Holland J H** (1975) *Adaptation Natural Artificial Systems* Ann Arbor: University Michigan Press.

[15] **Kaplan Jared,** McCandlish Sam, Henighan Tom, Brown Tom B, Chess Benjamin, Child Rewon, Gray Scott, Radford Alec, Wu Jeffrey, Amodei Dario (2020) Scaling laws for neural language models. *arXiv* https://doi.org/10.48550/arxiv.2001.08361

[16] **Kerstjens Stan,** Michel Gabriela, Douglas Rodney J (2022) Constructive connectomics: How neuronal axons get from here to there using gene-expression maps derived from their family trees. *PLoS Comput. Biol* **18**:e1010382 https://doi.org/10.1371/journal.pcbi.1010382 | PubMed

[17] **Kingma Diederik P,** Ba Jimmy (2014) Adam: A method for stochastic optimization. *arXiv* https://doi.org/10.48550/arxiv.1412.6980

[18] **Koonin Eugene V,** Wolf Yuri I (2009) Is evolution darwinian or/and lamarckian?. *Biol. Direct* **4**:42 https://doi.org/10.1186/1745-6150-4-42 | PubMed

[19] **McClelland James L,** Rogers Timothy T (2003) The parallel distributed processing approach to semantic cognition. *Nat. Rev. Neurosci* **4**:310-322 https://doi.org/10.1038/nrn1076 | PubMed

[20] **Meng Julia L,** Heckscher Ellie S (2021) Development of motor circuits: From neuronal stem cells and neuronal diversity to motor circuit assembly. *Curr. Top. Dev. Biol* **142**:409-442 https://doi.org/10.1016/bs.ctdb.2020.11.010 | PubMed

[21] **Miikkulainen Risto** (2025) Neuroevolution insights into biological neural computation. *Science* **387**:eadp7478 https://doi.org/10.1126/science.adp7478 | PubMed

[22] **Morgan C L** (1896) On modification and variation. *Science* **4**:733-740 https://doi.org/10.1126/science.4.99.733

[23] **Paszke Adam,** Gross Sam, Massa Francisco, Lerer Adam, Bradbury James, Chanan Gregory, Killeen Trevor, Lin Zeming, Gimelshein Natalia, Antiga Luca, *et al.* (2019) PyTorch: An imperative style, high-performance deep learning library. *arXiv* https://doi.org/10.48550/arxiv.1912.01703

[24] **Rumelhart David E,** Hinton Geoffrey E, Williams Ronald J (1986) Learning representations by back-propagating errors. *Nature* **323**:533-536 https://doi.org/10.1038/323533a0

[25] **Salimans Tim,** Ho Jonathan, Chen Xi, Sidor Szymon, Sutskever Ilya (2017) Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* https://doi.org/10.48550/arxiv.1703.03864

[26] **Saxe Andrew M,** McClelland James L, Ganguli Surya (2019) A mathematical theory of semantic development in deep neural networks. *Proc. Natl. Acad. Sci. U. S. A* **116**:11537-11546 https://doi.org/10.1073/pnas.1820226116 | PubMed

[27] **Shuvaev Sergey,** Lachi Divyansha, Koulakov Alexei, Zador Anthony (2024) Encoding innate ability through a genomic bottleneck. *Proc. Natl. Acad. Sci. U. S. A* **121**:e2409160121 https://doi.org/10.1073/pnas.2409160121 | PubMed

[28] **Stanley Kenneth O,** Clune Jeff, Lehman Joel, Miikkulainen Risto (2019) Designing neural networks through neuroevolution. *Nat. Mach. Intell* **1**:24-35 https://doi.org/10.1038/s42256-018-0006-z

[29] **Stanley Kenneth O,** Miikkulainen Risto (2002) Evolving neural networks through augmenting topologies. *Evol Comput* **10**:99-127 https://doi.org/10.1162/106365602320169811 | PubMed

[30] **Such Felipe Petroski,** Madhavan Vashisht, Conti Edoardo, Lehman Joel, Stanley Kenneth O, Clune Jeff (2017) Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv* https://doi.org/10.48550/arxiv.1712.06567

[31] **Sutton Richard S,** Barto Andrew G (1998) *Reinforcement Learning* Cambridge, MA: Bradford Books.

[32] **Todorov Emanuel,** Erez Tom, Tassa Yuval (2012) MuJoCo: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE. pp. 5026-5033 https://doi.org/10.1109/iros.2012.6386109

[33] **Towers Mark,** Kwiatkowski Ariel, Terry Jordan, Balis John U, Cola Gianluca De, Deleu Tristan, Goulão Manuel, Kallinteris Andreas, Krimmel Markus, Kg Arjun, *et al.* (2024) Gymnasium: A standard interface for reinforcement learning environments. *arXiv* https://doi.org/10.48550/arxiv.2407.17032

[34] **Wei Dongyu,** Talwar Vaishali, Lin Dayu (2021) Neural circuits of social behaviors: Innate yet flexible. *Neuron* **109**:1600-1620 https://doi.org/10.1016/j.neuron.2021.02.012 | PubMed

[35] **Williams Ronald J** (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* **8**:229-256 https://doi.org/10.1007/bf00992696

[36] **Xiaodong Cui,** Wei Zhang, Zoltán Tüske, Michael Picheny (2018) Evolutionary stochastic gradient descent for optimization of deep neural networks. *arXiv* https://doi.org/10.48550/arXiv.1810.06773

[37] **Yao Xin** (1999) Evolving artificial neural networks. *Proc. IEEE Inst. Electr. Electron. Eng* **87**:1423-1447 https://doi.org/10.1109/5.784219

[38] **Yoo Juyoun,** Dombrovski Mark, Mirshahidi Parmis, Nern Aljoscha, LoCascio Samuel A, Zipursky S Lawrence, Kurmangaliyev Yerbol Z (2023) Brain wiring determinants uncovered by integrating connectomes and transcriptomes. *Curr. Biol* **33**:3998–4005.e6 https://doi.org/10.1016/j.cub.2023.08.020 | PubMed

[39] **Zador Anthony M** (2019) A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun* **10**:3770 https://doi.org/10.1038/s41467-019-11786-6 | PubMed

[40] **Özel Mehmet Neset,** Gibbs Claudia Skok, Holguera Isabel, Soliman Mennah, Bonneau Richard, Desplan Claude (2022) Coordinated control of neuronal differentiation and wiring by sustained transcription factors. *Science* **378**:eadd1884 https://doi.org/10.1126/science.add1884 | PubMed

Peer reviews

Reviewer #1 (Public review):

In this article, the authors set out to understand how evolutionary selection could introduce structural priors into neural networks that act as an inductive bias to accelerate learning. To do so, the authors propose an evolutionary conditioning (ED) algorithm and analyse its properties. I find the conceptual framing of the paper very interesting, and it addresses an important question. Since the paper adopts a mostly theoretical approach with no comparison to empirical biological data, I do have a couple of concerns regarding the setup of the computational framework/method, which I think is incomplete and limits how much we can conclude from the current results.

Major Concerns:

(1) The evolutionary conditioning (EC) algorithm proposed works as fine-tuning training, plus propagation of the best parent network's weights to the next generation with added Gaussian noise. Conceptually, I find this to be a fairly implausible mechanism for evolution, since it requires carrying the entire set of network weights at some precision. The authors themselves point out that direct weight transfer could be problematic in the introduction.

(2) More importantly, I would like the authors to conduct a baseline / null model comparison, in which the "evolution" process consists simply of training a neural network for a small number of iterations, adding Gaussian noise, and repeating. The resulting network at each step serves as the "generations", which is then trained further. The same learning speed and dynamics analyses should be applied to this null model. What I am getting at is that I am not sure to what extent the EC algorithm can be thought of as "running a few iterations of SGD"

and chaining them together; how much work is the selection process in the GA actually doing?

(3) I am also unclear on why the EC algorithm does not improve throughout learning. Is this behavior the result of applying only a small number of fine-tuning steps? Presumably, with longer fine-tuning, the individual networks in the middle generations would also improve in performance?

(4) The EC algorithm applied to a single problem seems somewhat artificial in its setup. I would conceptualize evolution as learning a prior that conditions the network for a range of survival-related tasks. A more realistic setup would apply EC to an ensemble of tasks and then examine its impact on learning a specific task afterwards.

https://doi.org/10.7554/eLife.111582.1.sa1

Reviewer #2 (Public review):

Summary:

This paper studies the interplay of evolutionary and in-lifetime learning. The authors develop a neural network model in which initial weight configurations evolve under selective pressure, while fitness is determined by the network's performance after a learning period. They show that such a network displays very distinct learning dynamics from those trained by either gradient descent or genetic algorithms alone: in particular, they do not learn the task, but they show evidence of learning-to-learn and unusual representational structure.

Strengths:

(1) The writing, figures, and presentation of ideas were clear.

(2) The question of how evolution on initial weights combines with learning from within-lifetime experience to structure a learning trajectory seems interesting.

(3) The analysis of existing experiments was well-done, highlighting that though these networks did not really learn, they show latent learning structure that makes the network perform better from less data.

(4) The interplay between Baldwin & learning dynamics seemed novel and interesting, presenting many attractive puzzles.

Weaknesses:

First, the authors point to an important distinction between performing evolutionary selection on the weights pre- or post- lifetime training, the latter of which is Lamarckian. They argue, correctly, that their model is interesting because it selects on the weight initialisation, unlike, for example, Shuvaev et al. However, my understanding is that a long line of papers beginning perhaps with Hinton & Nowlan also do non-Lamarckian evolution: Hinton & Nowlan have unspecified weights (denoted '?' in the paper) that can be inherited and then learnt. Is this not exactly inheritance of initial conditions (in this case, whether learnable or not)? This novelty is a primary motivation of the paper, whereas to me it seems it was already apparent in Hinton & Nowlan, and developed further in what seems to be a long line of uncited literature (see next paragraph). As such, this paper's conclusions seem poorly positioned within the existing state of knowledge/literature.

Second, the algorithm is framed as novel, but I think it is a rediscovery. This framework is very close to MAML, in which an initial weight configuration is optimised by gradient descent to be good after a few steps of fine-tuning (Finn et al., 2017). The authors' approach differs in using a genetic algorithm to perform the training of the initial weights, avoiding some of the

computational complexities of MAML, especially after long fine-tuning. In this, the authors have, I think, rediscovered ES-MAML, MAML where the inner optimisation loop is gradient descent, while the outer is genetic (Song et al., 2020). Other similar work is "Meta-Learning by the Baldwin Effect" (Fernando et al., 2018).

Further, within Fernando et al. there is a rich literature review, almost none of which are cited by the authors. I point especially to Keesing & Stork, 1990, which appears to show a strong dependence of Baldwin-like improvements on the amount of data, something this paper also shows but explores less thoroughly.

To summarise my critique thus far: I think the literature already answers the main concern of the motivation (i.e. non-Lamarckian neural network evolution and learning), I think it has already discovered this particular algorithm, and I think past work has more thoroughly analysed behaviours similar to those presented in this paper. Without positioning correctly within this literature, the more general contribution of the paper is hard to establish. The true novelty of the authors' analysis seems to be the emphasis on Saxe et al.-like learning dynamics and its interplay with the Baldwin effect, but I am not certain of this without knowing the literature better.

Regarding experiments, there was an interesting effect where the EC networks didn't learn but did show latent learning (Figure 2, Figure 3), which sped up later learning (Figure 4). There were a few details I was surprised by on which I would appreciate clarity:

(1) The main result has basically no headline learning under EC. This will clearly be very dependent on parameters (e.g. if you add or remove enough training steps, the algorithm becomes SGD/GA, which both show learning). It seems like a natural analysis would examine this (e.g. a plot of final performance of EC after 4000 generations with different per-generation learning budgets).

(2) It is then shown that after 4000 generations EC can learn very quickly to perform the semantic task perfectly, at least within 200 generations (Figure 4C, and perhaps much sooner, Figure 4D, Figure 4F last panel; it was hard to say. This and the previous point seem somewhat inconsistent; was it just that Figure 3 used only 100 fine-tuning steps while the perfect-task-performing networks in Figure 4 required somewhere between 100 and 200? This seems to point to extreme parameter dependence. More broadly, how should I square this inconsistency/near-inconsistency?

(3) Figure 3k, and especially Figure 4f bottom right panel, seem to show networks that correctly separate all stimuli but cannot classify them. Should I understand this as networks learning to just push apart all pairs of datapoints without structure?

(4) If I understood the genetic algorithm correctly, only three individuals from each population seeded the next generation. This seems another important parameter to tune, since I think it is far lower than standard evolutionary work, but I am not sure.

Finally, the paper most interested me as a neural network learning puzzle: how can the network perform so badly, yet lead to such different post-fine-tuning results? The paper pointed to these as 'distinct' learning phenomena without explaining what was causing those differences. The only way I could square these results in my head was as above: that the 4000 generations pushed the initial representation to represent all datapoints differntly, effectively changing the learning problem gradient descent faces from one with a lot of structure (the semantic task) that leads to stepwise learning, to one in which it was basically linear regression on a set of well separated stimuli without the structure necessary for stepwise learning. Since the paper focuses so much on learning dynamics, and studies a task where such things can be precisely probed, it would have been nice to pin down exactly what was happening slightly more.

https://doi.org/10.7554/eLife.111582.1.sa0