ENCODING INNATE ABILITY THROUGH A GENOMIC BOTTLENECK

Alexei Koulakov
Cold Spring Harbor Laboratory
Cold Spring Harbor, NY 11724
koulakov@cshl.edu

Sergey Shuvaev
Cold Spring Harbor Laboratory
Cold Spring Harbor, NY 11724
sshuvaev@cshl.edu

Anthony Zador
Cold Spring Harbor Laboratory
Cold Spring Harbor, NY 11724
zador@cshl.edu

March 16, 2021

ABSTRACT

Animals are born with extensive innate behavioral capabilities, which arise from neural circuits encoded in the genome. However, the information capacity of the genome is orders of magnitude smaller than that needed to specify the connectivity of an arbitrary brain circuit, indicating that the rules encoding circuit formation must fit through a "genomic bottleneck" as they pass from one generation to the next. Here we formulate the problem of innate behavioral capacity in the context of artificial neural networks in terms of lossy compression of the weight matrix. We find that several standard network architectures can be compressed by several orders of magnitude, yielding pre-training performance that can approach that of the fully-trained network. Interestingly, for complex but not for simple test problems, the genomic bottleneck algorithm also captures essential features of the circuit, leading to enhanced transfer learning to novel data sets. Our results suggest that compressing a neural circuit through the genomic bottleneck serves as a regularizer, enabling evolution to select simple circuits that can be readily adapted to important real-world tasks. The genomic bottleneck also suggests how innate priors can complement conventional approaches to learning in designing algorithms for artificial intelligence.

Introduction

Many animals are born with impressive and elaborate behavioral capacities. Soon after birth, a spider can build a web, a whale can swim, and a monkey fears snakes. From an evolutionary perspective, it is easy to see why such innate abilities would be selected for: Those individuals that can survive beyond their most vulnerable early hours, days, or weeks are more likely to survive until reproductive age and hence produce progeny at a higher rate. Of course, in practice there is no crisp distinction between innate and learned abilities; innate abilities form a foundation for learning, and animal behavior arises from an interaction between these two processes. Although learning has been studied extensively in the context of artificial intelligence, there has been much less theoretical attention devoted to the structure of innate behaviors.

Innate behaviors are encoded in the genome and can be expressed in the neural circuits already present at birth. However, this poses a challenge: How can a complex neuronal connectivity diagram be encoded into a genome? The size of the genome provides an approximate upper bound on the amount of information transmitted from generation to generation. The genome of the simple worm *C. elegans* is about 10^8 base pairs (Bargmann, 1998), so it could transmit up to about 2×10^8 bits. This in principle would be more than adequate to explicitly encode the highly stereotyped connectivity among the 302 neurons in the *C. elegans* brain, since even a dense 302^2 connection matrix would take at most 9×10^4 bits to store. On the other hand, the human genome is only about an order of magnitude larger than that of *C. elegans*

($\sim 10^9$ bits), whereas the human cortex has about 10^{10} neurons, so the (sparse) cortical connectivity matrix might require at least 10^{15} bits to specify. This implies that the human cortex would require about 5 orders of magnitude more information to specify explicitly than is available in the genome (Zador, 2019). Since the genome encodes the blueprint for wiring up the nervous system, a natural question is how the small amount of information contained in the genome can instruct the creation of the large capacity cortex. We refer to the long recognized (Tessier-Lavigne and Goodman, 1996; Sperry, 1963; Stanley et al., 2019) mismatch between the information capacity of the genome and the complexity of the resulting neural circuit as a "genomic bottleneck" (Zador, 2019).

The mismatch described by genomic bottleneck implies that the connectivity of most neuronal circuits, including the mammalian cortex, is not explicitly specified, neuron-by-neuron, in the genome. Rather, the genome specifies rules for connectivity. It is straightforward to formulate simple, low-complexity rules that specify the connectivity in arbitrarily large networks. For example, the simple rule "connect to your four nearest neighbors" specifies a grid of potentially unlimited size (Fig. 1A). Another class of rules specifies connections between cells based on the surface markers they express (Sperry, 1963; Zipursky and Sanes, 2010; Wei et al., 2013) (Fig. 1B); axons can exploit these markers to find their destinations (Goodhill and Baier, 1998). The columnar organization that is observed in many brain regions allows for the replication of similar connectivity modules throughout the brain, thus limiting the number of parameters needed to wire the circuit (Itzkovitz et al., 2008). Complex structures such as orientation columns in the visual cortex can be induced to self-organize from simple use-dependent rules (Von der Malsburg, 1973). Developmental rules such as these can dramatically reduce the amount of information needed to specify the connectivity of a neural circuit, prior to extensive experience.

In spite of this extensive literature on modeling development, these simple developmental rules do not typically specify networks with the capacity to perform complex general computations. Thus, although such rules can readily specify the formation of repeated modules that enable the emergence of receptive fields in the visual cortex (Linsker, 1986), such stereotyped modules cannot directly encode more specialized knowledge like a spider's capacity to build a web or a rat's innate fear of fox odor (LeDoux, 2012). We therefore set out to explore how low-complexity rules can give rise to networks that perform complex well-defined computations. Within the framework of artificial neural networks (ANNs), we seek to compress the complex connectivity (weight matrix) into a much smaller "genome." The decoding of this genome into the initial weights of the network enables the network to perform well upon initialization, without additional training. This decoding is analogous to the neurodevelopmental processes by which the genome provides a blueprint for circuits that enable animals to perform essential tasks at or soon after birth. We hypothesized that under some conditions, compressing the weight matrix through a "genomic bottleneck" would extract the most useful and important features of the connectivity; the genome would act as an "information bottleneck" (Tishby et al., 2000). In this way, a physical constraint—the limited size of the genome—might actually be an algorithmic advantage, serving as a regularizer and thereby turning a potential "bug" into a feature.

Implementation of the Genomic Bottleneck

To test these ideas, we trained standard feedforward ANNs on well-studied supervised learning tasks. ANNs consist of nodes ("neurons") connected by weights ("synapses"). Artificial neural networks are typically initiated randomly—*tabula rasa*—and acquire their functionality through learning. When an ANN learns a task, the "knowledge" of the task is summarized in the weights of the ANN. We use "connectivity" to describe both the specification of which connections are non-zero, as well as the strengths or weights of those connections. We refer to the trained network as the "phenotype network," or "p-network" (Stanley et al., 2009).

We sought to compress the p-network through a genomic bottleneck, preserving as much of the performance as possible. The compressed network serves to initialize the network, endowing it with innate abilities prior to learning. To search widely over the space of possible compressions, we used a separate ANN—a "genomic" or "g-network"—to generate the p-network (Fig. 1C). The formulation of the neurodevelopmental process as an ANN allows us to focus on the genomic bottleneck at a conceptual level, without the need to model the complexities of neural development. The inputs to the g-network are a pre- and postsynaptic pair of neurons, and the output of the g-network is the expected strength of the connection between these neurons. For sparsely connected networks, the strength will often be zero. This formulation is inspired by neurodevelopmental rules based on local pair-wise interactions between neurons (Sperry, 1963; Zipursky and Sanes, 2010), but it is not intended as realistic model of neural development (Stanley et al., 2009; Barabási and Barabási, 2020). To facilitate the efficient search for g-networks that achieve good compression, we used stochastic gradient descent (Ha et al., 2016) rather than evolutionary algorithms (Stanley et al., 2009, 2019) to achieve end-to-end optimization of both the g-network and the p-network.

We represented each neuron in the p-network by a unique binary label. Each binary digit in this label can be interpreted as an indicator of the presence of one type of a "molecular tag" in the expression profile of this neuron. Connectivity

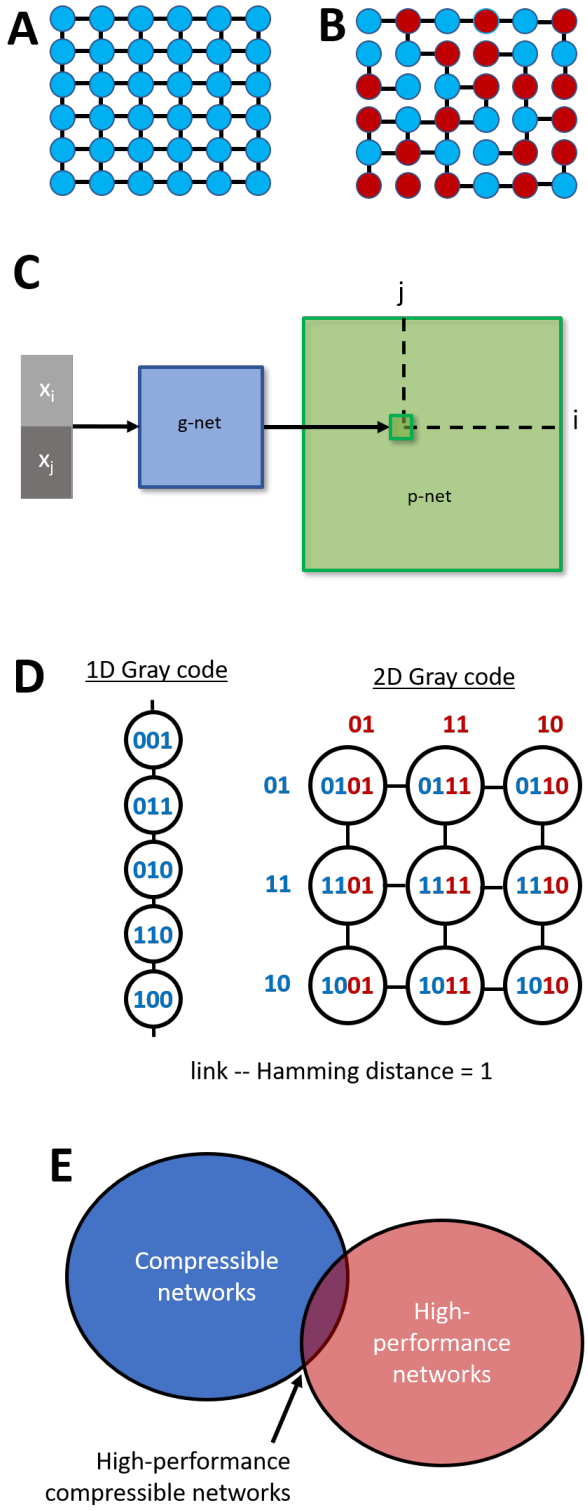

Figure 1: Simple rules can specify networks. (A) A very simple nearest neighbor wiring rule. (B) A somewhat more complex rule (only connect to nearest neighbors of opposite color) leads to a more complex network. (C) Network specification through a genomic bottleneck. The input to the genomic network ("g-network") is a pair of neurons specified by binary strings. Each neuron has a unique label. The two binary labels are passed through the g-network, which assigns the strength of the connection between the two neurons in the "p-network." Because the number of parameters in the g-network is smaller than the number of parameters in the p-network, the g-network is effectively compressing the p-network. (D) Neuron labels are assigned by a 2D Gray code, so that neurons close in image space have similar (small Hamming distance) labels. (E) G-networks seek to discover p-networks that both solve the problem well and are compressible.

is effectively guided by interactions of pairs of molecules expressed on the pre- and postsynaptic membranes. The g-network framework can naturally accommodate the addition of new neurons, new layers, and new connections between layers. A network with N neurons requires $log_2(N)$ binary tags to specify each neuron uniquely, so the input to the g-network is $log_2(N)$ units (binary digits) for each neuron. Because the size of the g-network's input grows only logarithmically as more neurons are added to the p-network, the g-network can implement connectivity of arbitrarily large p-networks without substantial changes in size. Thus, the small amount of information parameterizing the g-network can be used to encode even very large p-networks.

Neuronal projections are often organized topographically. For example, nearby ganglion neurons in the retina encoding nearby points in physical space project topographically via the thalamus to nearby points in visual area V1. However, we did not explicitly encode geometric space into the network (Stanley et al., 2009). Instead, we adopted a more general approach in which each neuron was labeled according to a two-dimensional Gray code (Frank, 1953), so that the binary strings of nearby neurons differ by only a small number of bits (Fig. 1D). The Gray's code facilitates the discovery of connectivity rules that exploit spatial structure, but also has the potential to allow the network to discover wiring motifs associated with distinct neuronal "types" or "classes". For example, one bit of the neuronal specification in the Gray's code could represent the distinction between excitatory and inhibitory neurons, and another bit could denote a particular subtype of inhibitory neuron. In this way, the characteristic connectivity patterns associated with specific neuronal subtypes in brains (Kepecs and Fishell, 2014) could readily arise in these ANNs.

Although, in principle, a sufficiently large g-network could perfectly recapitulate the p-network by "memorizing" all of the connections exactly, this would fail to compress the p-network, and thus would be unlikely to extract more general wiring motifs. We therefore focused on a regime where the size (complexity) of the g-network is substantially smaller than the size of the p-network, encouraging the g-network to discover compact wiring rules. Our goal is to discover network architectures that are both high performance and compressible (Fig. 1E). This setting can be viewed as a lossy compression problem, where the success of compression is measured not by e.g. the reconstruction error as in typical lossy image compression, but rather by the ability of the uncompressed weight matrix to perform well on the target task without further training.

Our approach can be described as the search for a solution that balances two competing goals. First, we wish to achieve good *ab initio* performance, i.e. we wish to minimize the error $E_0(G)$, where $E_0(G)$ is the error before training of the p-network encoded by the corresponding g-network G. Second, we also wish to limit the complexity of the genomic network, specifically the number of parameters or entropy $H(G)$ of the genome G. We thus formulate our overall goal as a lossy compression problem in which we seek to minimize an objective function J with respect to the genome parameters G, subject to a constraint:

$$J = E_0(G) + \lambda H(G), \tag{1}$$

where λ is a positive parameter that specifies the tradeoff between the two goals. In this formulation, the second term can be seen as a regularizer, related to techniques such as weight pruning (LeCun et al., 1990; Han et al., 2015), that seek to keep the weight matrix simple.

We used an iterative algorithm to find g-networks which generate p-networks that are both compressible and achieve good ab initio performance (see Methods). First, we used standard stochastic gradient descent to find a weight matrix $W^{(1)}$ of a p-network that minimizes the training error. The elements of this weight matrix are then used to train the weights of a g-network. Specifically, the training set for the g-network consists inputs $\{b_i, b_j\}$ and target outputs $w_{ij}^{(1)}$, where b_i is the binary representation of the ith unit of the p-network, and $w_{ij}^{(1)}$ is the connection from unit i to unit j. The g-network trained in this way thus generates an approximation $\hat{W}^{(1)}$ of the p-network weight matrix on which it was trained. This approximation is then used as the initialization to train the next iteration $W^{(2)}$ of the weight matrix, and the process is iterated to yield $\hat{W}^{(2)} \dots \hat{W}^{(K)}$. This procedure yields networks that are both capable of accurate initial (innate) performance and occupy limited amounts of space in the genome (compressible).

Solving MNIST

We first tested our approach on a classic supervised learning problem, handwritten digit recognition. In this problem, a network is trained with examples of handwritten digits (0-9) taken from the MNIST dataset (Fig. 2A; *see Methods*), and learns to assign labels to new examples of these digits it has not previously encountered. For the p-network we used a standard fully-connected network architecture with 784 pixel units at the input layer and one hidden layer with 800 units, for a total of 6×10^5 parameters. With random initial weights, the performance rose from random (10%) to about 98% (Fig. 2C) after about 20 epochs.

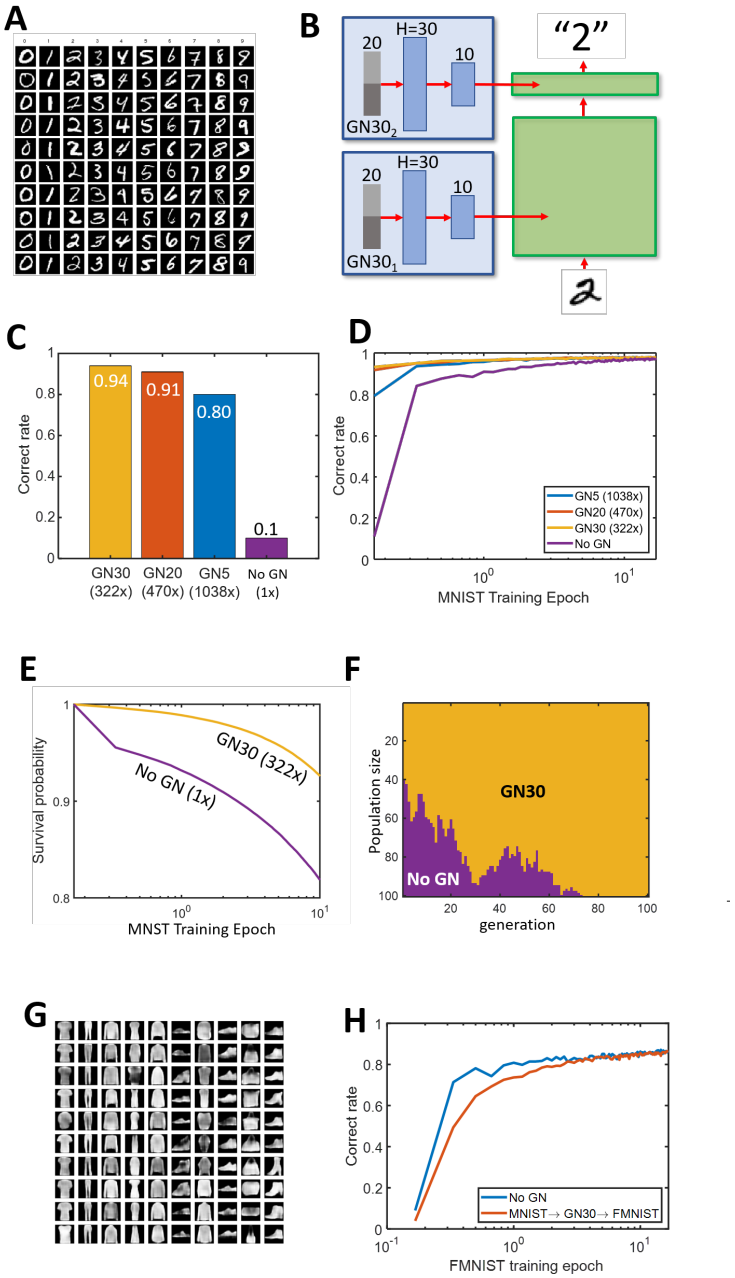

Figure 2: Genomic bottleneck solution to MNIST. (A) Examples of MNIST data set of handwritten digits. (B) We used the two layer fully connected MNIST network created by individual g-networks in each layer (C) p-network achieves excellent *tabula rasa* performance even with 1000-fold compression. (D) Tradeoff between compressibility and initial performance. (E) Fitness advantage of model organism with high *ab initio* performance. (F) Model organism with high *ab initio* performance dominates population. (G) Example of fashion-MNIST data set. (H) Failure of transfer learning f-MNIST data set.

We then used a much smaller g-network, GN30, with only about 2×10^3 parameters to compress the p-network. The g-network trained using the algorithm described above generated a p-network with 94% correct performance upon initialization. Thus the g-network was able to achieve 322-fold compression with *ab initio* performance almost equal to that of the fully trained network. We observed a tradeoff between the degree of compression and the *ab initio* performance (Fig. 2C, D), but good *ab initio* performance of 79% correct could still be achieved using GN5, a network with 1038-fold compression. These results demonstrate that there exist p-networks that are both compressible and high-performance.

To illustrate how high *ab initio* performance could provide an evolutionary advantage, we formulated a simple model in which an organism's survival to reproductive maturity—its fitness—was proportional to its performance on this task. In this model, the probability $p(t)$ that the organism is still alive at time t is given by

$$p(t+1) = p(t)[1 - \alpha(1 - c(t))]. \tag{2}$$

Here, $c(t)$ is the rate of correct performance at time t and $0 < \alpha < 1$ is a parameter that determines the contribution of this trait to survival. This model thus relates correct performance on the task to survival (Fig. 2E). Fig. 2F shows how, over successive generations, the fraction of individuals with high *ab initio* performance increases at the expense of individuals which rely solely on learning to acquire fitness. As expected, over several dozen generations the individuals with higher *ab initio* fitness dominate the population, completely supplanting those initialized *tabula rasa*, which must learn everything from the environment. Many factors in the real world could serve to complicate this simple model, which for example does not explain the prolonged period of posnatal helplessness of mammals. Nonetheless, the model provides an intuition for why evolution might be expected to maximize high *ab initio* performance.

We hypothesized that passing the wiring diagram through the genomic bottleneck would extract the most useful and important features of the connectivity and enable generalization to related tasks. To test this idea, we used the related "fashion" MNIST, or F-MNIST dataset, which has the same format as the MNIST data set but consists of ten different categories of clothing (shirts, shoes, etc; Fig. 2F). Disappointingly, we observed no enhancement of F-MNIST learning upon initializing weights using MNIST-trained g-network. Indeed, the p-network adapted from MNIST actually showed somewhat slower learning than a naive network (Fig. 2G), as though the network first had to unlearn MNIST before learning F-MNIST. We hypothesized that this failure to generalize across visual recognition tasks was due to overfitting on the specifics of MNIST dataset, due to the relative simplicity of the tasks and the network used to solve them: Both of these tasks can solved by relatively shallow (2-layer) "specialist" networks, leaving minimal knowledge available to be shared across two tasks.

Solving Cifar10

To test whether there are conditions where the genomic bottleneck might extract features that generalize over multiple datasets, we applied the algorithm to a more complex problem that requires a deeper network. We used the CIFAR10 dataset, which consists of 60k color images drawn from 10 categories such as airplanes, cars, and cats (Fig. 3A). For the p-network we used a standard 9-layer convolutional neural network (CNN) architecture with about 1.4×10^6 weights (*see Methods*). We compressed each CNN layer of the p-net with a separate g-network (Fig. 3B). Similarly to MNIST, the compressed CIFAR10 network reached high performance. GN50, a network with 92-fold compression, achieved initial performance of 76% (vs. naive 10%), fairly close to the fully trained performance of 89% (Fig. 3C, D). Thus, as with MNIST, the g-network achieved approximately two orders of magnitude compression while maintaining good initial performance.

The learning dynamics (Fig. 3D) demonstrate the utility of genomic compression for achieving enhanced initial performance. However, it was not clear whether this was also associated with faster learning. To our surprise, genomic compression had no effect on the learning trajectory; the only speedup was due to the higher initial performance (Fig. 3E). Thus genomic compression, at least under these conditions, did not affect the learning rate.

To assess whether the structure that g-networks extracted from the CIFAR10 dataset could be useful for other datasets, we tested transfer from the CIFAR10-trained network to a related problem. We used the Street View House Numbers (SVHN) dataset, which contains images of street numbers in a format similar to the CIFAR10 dataset (Fig. 3F). We first confirmed the effectiveness of a standard algorithm for transfer learning. We trained the p-network on the CIFAR10 dataset without compression, and then used the p-network's weights as a starting point for SVHN training (CIFAR10 transfer). As expected, this procedure accelerated learning, reducing training time from about 20 training epochs (Fig. 3G, blue line, "No GN") to a single epoch (Fig. 3G, "CIFAR10 L1-9"). This implies that CIFAR10 dataset contains features that are similar to SVHN dataset.

We next compared the standard transfer learning algorithm to an algorithm based on the genomic bottleneck. To achieve transfer with the genomic bottleneck, we used g-networks to generate p-network as described above and then used this

p-network as an initial condition for SVHN training (g-network mediated transfer). Remarkably, the performance of g-network mediated transfer was indistinguishable from the standard approach (Fig. 3F, orange solid line, "L1-9"), even though in this case the number of transferred parameters was 92 times fewer. These results indicate that whatever is crucial for transfer from CIFAR10 to SVHN is captured by the nearly 92-fold smaller g-network.

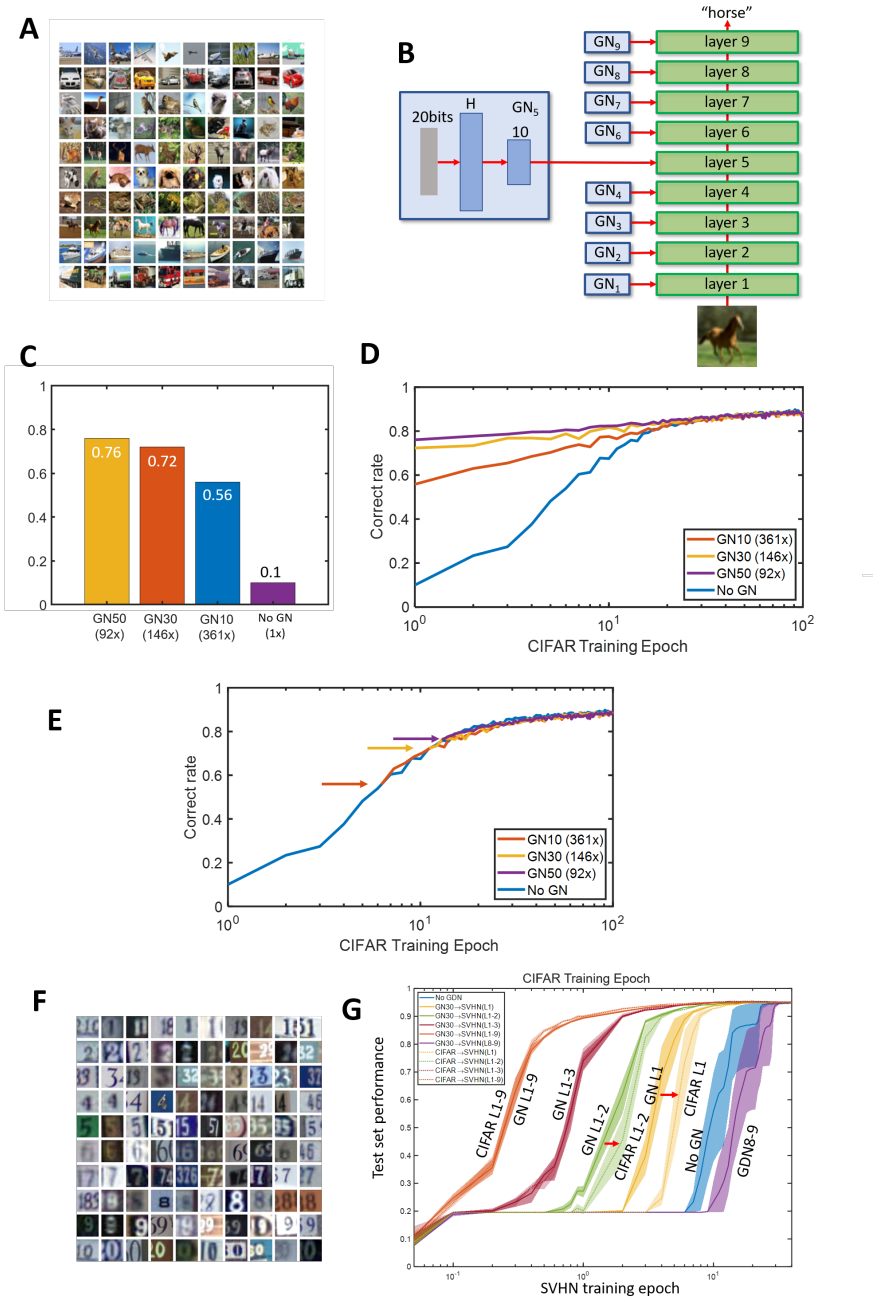

Figure 3: Genomic bottleneck solution to CIFAR10. (A) Examples of CIFAR10 data set of images. (B) To classify CIFAR10 images, we used nine layer all convolutional network. Each layer was created by individual g-networks. (C) p-network achieves excellent *tabula rasa* performance even with 100-fold compression. (D) Tradeoff between compressibility and performance. (E) Learning rate of compressed networks is the same as *tabula rasa* networks. (F) Example of SVHN data set. (G) Transfer learning to SVHN data set.

To further dissect the consequences of genomic compression, we examined the effect of transferring only one or a few layers at a time. When only the first two lower layers (layers 1-2) were transferred from CIFAR to SVHN, while

randomly initializing the remaining layers, genomic transfer actually yielded faster learning than direct transfer (Fig. 3F, red arrows). For example, when layer 1 was initialized with g-network, 50% performance was reached in about 3.2 training epochs, while similar levels were only achieved in 4.8 epochs using direct CIFAR10 transfer—a 1.5-fold difference. Thus, it appears that the lower layers of the network contain features that generalize across datasets, and these features are extracted particularly well using the g-network based compression algorithm. On the other hand, transferring the last two layers of the network resulted in slower training compared to naive case, a result reminiscent of our previous result with MNIST-to-F-MNIST transfer (Fig. 2H). This result implies that the last two layers of CIFAR10-trained network contained features that are specific to the dataset and were not useful for the recognition of the house numbers in SVHN data.

Taken together, these findings demonstrate that g-networks can extract structure that is generalizable across datasets. Compression with g-networks yields performance that is comparable to—and in some cases better—than simple uncompressed weight transfer, indicating that g-networks identify a special subclass of p-networks that are compressible and capture essential structure of the data (Fig. 1D). This enhancement is particularly evident is the rate transfer of the lower layers in deep nets (Fig. 3G). Interestingly, the receptive fields of neurons in the lower visual system show substantial similarities between different species, while higher layers are more specialized (Rodieck and Rodieck, 1998). This parallel with our results suggests that the early visual system may have extracted a simple yet potent set of features while subject to genomic bottleneck-like constraint.

Discussion

We have proposed that a genomic bottleneck arises inevitably because of the need for a relatively low-capacity genome to specify the complex neural circuits required for innate behaviors. We argue that under a wide range of conditions, there is evolutionary pressure for organisms to be born with as much innate ability as possible, and thereby to maximize their fitness at birth (Fig. 2E). This leads to a model in which genomes, and the circuits they encode, are co-optimized in nested loops: an inner loop corresponding to "learning" in animals, and an outer loop corresponding to "evolution." Our results suggest that dividing the usual machine learning problem into such nested loops linked by a low-information bottleneck serves as a regularizer on the resulting neural circuit, guiding it to find simple circuit motifs that can be reused and can adapt with changes in the environment.

The genomic bottleneck can be viewed as a constraint forcing lossy compression of the weight matrix (Eq. 1). The idea of minimizing the description length, or Kolmogorov Complexity, of the weights has been proposed before (Hinton and Van Camp, 1993). Although the genomic bottleneck algorithm was motivated by considerations of the relative size of the genome and the connectome, it has close parallels with the "information bottleneck" method (Tishby et al., 2000). We hypothesize that, by squeezing the neural circuit diagram through a much smaller genome, evolution has extracted the most useful and important network motifs.

To perform tasks, the compressed (genomic) representation must be uncompressed into a functional network through a process analogous to neural development. For reasons of efficiency, we have used gradients to optimize both the inner and outer loops. Evolution, which can be viewed as a form of reinforcement learning but without a gradient, is in general a relatively slow and inefficient algorithm, successful because it operates on massive numbers of individuals in parallel over hundreds of millions of years. The feedback in our algorithm that guides the gradient from each generation—the fact that the trained weight matrix in the kth generation is used to modify the genome in the $(k+1)^{st}$ generation—can be viewed as a form of Lamarckian evolution, and is, as such, biologically unrealistic. The net effect of our approach, however, is similar to Darwinian evolution our algorithm is designed to optimize fitness. Our algorithm can be seen as an implementation of the Baldwin effect (Baldwin, 1896; Hinton and Nowlan, 1987), according to which, if the ability to learn a particular behavior rapidly conferred a selective advantage, that ability would, over the course of evolution, be "genetically assimilated" and might appear to have arisen through a Lamarckian process.

Network pruning can be viewed as a method of network compression, complementary to the genomic bottleneck mechanism considered here (LeCun et al., 1990; Han et al., 2015). Convolutional neural networks (LeCun and Bengio, 1995) represent an example of network pruning whereby each neuron only connects to a small fraction of other neurons in lower layers. Another example is provided by the lottery ticket hypothesis, according to which the number of weights in a well-performing network can be substantially reduced by discovering "winning ticket" sparse subnetworks (Frankle and Carbin, 2018). Cortical networks are inherently sparse with each neuron connecting to a small fraction of about 10^-6 of other cortical cells. Evolution is thought to select sparse and functionally important connections due to physical constraints, such as a space and time limitations(Chklovskii et al., 2002). Even after sparsification, cortical connections cannot be encoded in the genome with the single neuron precision. Thus, pruned connectivity is a default solution to the evolution of cortical fitness, and does not by itself resolve the discrepancy between cortical and genomic information

capacities; even sparse connectivity must be further compressed through the genomic bottleneck. Here we study the additional rules that can encode both fully and sparsely connected networks.

We developed a genomic bottleneck algorithm that could achieve several orders of magnitude compression on two standard ANN tasks, MNIST and CIFAR. Although it might seem surprising that these networks could be so highly compressed with relatively little loss of performance, there is a considerable literature on network compression (Choudhary et al., 2020).

For example, a standard technique—weight pruning—can eliminate 90% of parameters with minimal loss of accuracy (LeCun et al., 1990; Han et al., 2015), but such pruned networks are typically difficult to train. Unlike sparse networks obtained by pruning, the genomic bottleneck approach described here leads to networks that are not only compressed but also easily trained. According to the lottery ticket hypothesis, the compressibility of pruned might be explained in part by the existence of "winning ticket" subnetworks (Frankle and Carbin, 2018). However, the "winning ticket" sub-network does not exhibit good *ab initio* performance; indeed, the winning ticket neurons actually tend to change their synaptic weights even more than the other neurons. Thus the relationship between winning ticket subnetworks and those obtained by the genomic bottleneck remains unclear.

Our results contribute to a growing literature highlighting the importance of inductive biases in machine learning. Much of this literature is focused on achieving faster learning. Perhaps the most successful examples are convolutional neural networks (LeCun and Bengio, 1995), which exploit the translational invariance of images with an architecture inspired by the structure of receptive fields in early sensory cortex (Hubel and Wiesel, 1962). However, the present work—inspired by evolutionary constraints (Fig. 2D)—is focused not on faster learning but rather enhanced initial performance, a goal that has received comparatively less attention (but see (Gaier and Ha, 2019)). Indeed, in our experiments we find that genome-initialized networks start off at a higher level of performance but then follow the same trajectory as randomly-initialized networks (Fig. 3D). Although these results highlight the potential dissociation of two distinct process—better initial performance and faster learning through inductive biases—there is likely strong evolutionary pressure to maximize both.

The relative importance of genomically-encoded innate structures in determining specific human abilities such as language has been hotly debated, but the importance of innate factors to the behavior of other animals is less controversial. For both humans and other animals, the better question is usually not whether a behavior is innate, but rather how innate and learned factors interact. For example, the propensity to form "place fields" in the hippocampus is innate—a map of space emerges when young rat pups explore an open environment outside the nest for the very first time (Langston et al., 2010)—but the content of place fields is learned, as new place fields form whenever the animal enters a new environment. In this example it appears that, as suggested by (Fig. 2D), *ab initio* performance has been maximized by providing a scaffolding for place fields to appear.

The genomic bottleneck takes its inspiration from fundamental considerations about the evolution and development of brain circuits. Although the genomic bottleneck algorithm builds on existing machine learning techniques, and yields surprisingly effective performance, we have not attempted to optimize this approach to compete with state-of-the-art benchmarks. The bottleneck framework is potentially quite rich, and could be extended in several directions. For example, we have not explored variations in the structure of the genomic network, *e.g.* by imposing a sparseness constraint. Such a constraint would have the physical interpretation of limiting interactions among surface neuronal markers. Similarly, at present, each layer is compressed with its own genome, but it would natural to attempt to extract regularities among layers by encoding them with a single genome. Furthermore, in the current formulation the decoding of the genomic network is deterministic, whereas developmental rules are often stochastic, so the decoding framework might be generalized to include rules like "connect on average to 10% of your nearby neighbors." Finally, the framework could be extended to co-optimize the learning rules and the wiring.

Acknowledgments We are grateful for many helpful comments from Saket Navlakha, Blake Richards, Tatiana Engel, Xaq Pitkow, Kaleb Vinehout, Barak Pearlmutter. We would like to acknowledge support from Deep Valley Labs.

Methods

MNIST/F-MNIST dataset

For MNIST and F-MNIST datasets, we used a fully connected 2-layer network that included 800 hidden layer ReLU units (Simard et al., 2003). We did not use data augmentation for simplicity and our network could be trained to 98% performance on testing data. The number of parameters in the MNIST network was therefore $28^2 \times 800 + 800 \times 10$ weights and $800 + 10$ biases amounting to the total of 636010. We used three g-networks to encoded two weight matrices and one bias vector for the hidden layer. The ten biases for the output layer were not compressed since the corresponding g-network would include more than ten parameters. The structures for various configurations of the three g-networks are listed in Table 1. The schematic of the structure of g-network for MNIST dataset is shown in Fig. S.1A.

Each neuron was described by a binary label of length 10. For the neurons in the input image, the label encoded two coordinates of the neuron's position in the image, 5 bits for the "X" and "Y" coordinates. Both coordinates were represented by the Gray code. The neurons in the hidden layer were represented by simple binary codes ranging from 1 to 800, since their order is of no particular importance. The ten neurons in the output layer were encoded by the one-hot vector of 10 bits. Each neuron in the networks was therefore described by a ten-bit label. The inputs into each of the two g-networks that generated weight matrices represented pairs of neurons and had the length of 20 bits. The output of g-network is the value of the corresponding weight between two input neurons and was a single real number (Table 1). For the network generating biases for the hidden layer, the input contained the binary label for the neuron and was therefore 10 bits long.

There are several options to train g-networks. The simplest one is to use end-to-end backpropagation from the dataset (MNIST) to g-network using automatic differentiation of the deep learning library (PyTorch, Fig. S.1A). We found this implementation to be inefficient as it involves generation of the entire set of weight matrices for each minibatch of MNIST images. Instead, we developed the intermittent training paradigm. Before the first iteration, the g-networks are initialized randomly. On each iteration of this method, we start with the g-networks generating weights of the MNIST network (p-net). We then train the p-net using a subset of images. For the MNIST network, this training used 10,000 images from the training set or 1/6th of the epoch. This yielded a higher performance p-net with the set of weights described by matrix W_n. We then train g-networks to approximate this weight matrix by backpropagating the difference between the g-network output ($\tilde{W}_n$) and W_n. We used different number of weights to train each of the three g-networks on each iteration (10^5, 10^4, and 10^4 for hidden layer, output layer, and hidden layer biases g-networks, respectively). This amounted to about 1/6th of all weights and further accelerated training in each generation. We then used the adjusted g-networks to generate the complete set of p-net weights $\tilde{W}_{n+1}$ that served as initial conditions for the next generation (Fig. S.1B). This set of iteration mimicked real biological evolution as it alternated the generation of p-nets from g-networks, analogous to the neural development, and improvement of p-nets similar to the natural selection. We repeated these iterations 500 times to achieve the asymptotic performance.

CIFAR10/SVHN datasets

In this example we used all convolutional 9-layer implementation of network (Springenberg et al., 2015) (Fig. S.1B). Between layers 3 and 4, we included the dropout layer with 50% dropout probability. The network could be trained to 89% correct rate without data augmentation. Each layer in this 9-layer CNN was generated via two g-networks, one for the weight matrix and one for the biases. To provide inputs into the g-networks for weights, we described positions in the weigh matrix by a 20-bit binary number. For the lowest 8 layers, the binary number was composed of 2+2 bits containing Gray code representation for the input coordinates within the CNN kernel, 8 bits representing the input filter type, and 8 bits representing the output filter type. The latter two components were formed as consecutive binary numbers, since input/output filter identity is not expected to form a continuous topographic space. In this representation, the input neurons were identified by a 12-bit binary label (2 + 2 Gray + 8 binary) while the output neurons are identified by the 8-bit label. The binary labels for the last CNN layer were composed of 1+1 binary Gray (dummy) bits representing two coordinates within the kernel, a 8-bit binary number representing the input filter, and

Table 1: The structure of g-network for MNIST dataset

g-network name	g-network structure	Bias net structure	g-network parameters	MNIST parameters	Compression
GN5	$(20-10-5-1)\times 2$		613		1038x
GN20	$(20-20-10-1)\times 2$	$10-5-1$	1,353	636,010	470x
GN30	$(20-30-10-1)\times 2$		1,973		322x

Table 2: The structure of g-network for CIFAR10 dataset

g-network name	g-network structure	Bias net structure	g-network parameters	CIFAR10 parameters	Compression
GN10	$(20-10-10-1)\times 9$		3,798		361x
GN30	$(20-30-10-1)\times 9$	$8-10-1$	9,378	1,369,738	146x
GN50	$(20-50-10-1)\times 9$		14,958		92x

a 10-bit one-hot vector encoding the output class. The bias networks for the first eight layers received an 8-bit label encoding the output filter. The bias network for the last layer received the one-hot vector encoding one of the ten output classes. The structures of different g-networks used are summarized in Table 2.

To train the 18 g-networks described above, we used the intermittent training strategy described for the MNIST network. We used minibatch sizes of 10, 100, and 1000 for bias g-networks for layers 1-3, 9, bias g-networks 4-8, and all weight matrix g-networks, respectively. We used the minibatch size of 128 to train the CIFAR10 network. We used SGD optimizer for CIFAR network with the learning rate of 0.05 and momentum of 0.9 for stability. We used Adam optimizer for all g-networks. In each iteration, we first trained CIFAR10 network using 10 complete epochs, i.e. 10 complete sweeps through the entire training set of images. In the second step, we trained weight and bias g-networks using 2 and 10 epochs respectively (we trained g-network weight networks for layer 1 and 9 using 12 epochs in each iteration) to match the CIFAR network adjusted weights resulting from the first step. This sequence of two steps was repeated 500 times.

Because our network was relatively deep (9 layers), we encountered a problem with initialization of g-networks. Indeed, if g-networks are initialized randomly, they produce p-nets that are far from the optimal fixed point. We found that such p-nets are impossible to train. This problem is exacerbated in moderately deep p-nets due to the exponential divergence of initialization errors from layer to layer. In practice, such p-nets return zero activations, which yield no gradients of weights. To solve this problem, we implemented the weight annealing strategy. In each iteration of our algorithm (out of 500), before the p-net was trained, the weight matrices and biases of the p-net were combined from the results of GIFAR training in the previous iteration, W_{n-1}, and the weights generated by g-network in the previous iteration, $\tilde{W}_{n-1}(G)$:

$$W_n = \epsilon(n) W_{n-1} + \tilde{W}_{n-1}(G)[1-\epsilon(n)] \tag{3}$$

The coefficient $\epsilon(n)$ determined the degree to which the inputs from g-networks affect the p-net's weight matrix. If $\epsilon(n) = 1$, the weight matrix of p-network is entirely determined by the result of previous iteration of CIFAR training and is not sensitive to the inputs from g-net. In the other extreme, when $\epsilon(n) = 0$, the values of p-net weights and biases are entirely generated by the g-networks. We therefore assumed that $\epsilon(n) \approx 1$ in the beginning of training, when g-networks are naive, and $\epsilon(n) \to 0$ in the end of training. We adopted an exponential annealing schedule with $\epsilon(n) = \exp(-n/\lambda)$, where parameter $\lambda = 20$ determined the number of iterations in the intermittent training over which the g-networks are assumed to be naive and irrelevant, and initialization using g-networks is assumed to be detrimental. Since, in our approach, the total number of iterations is 500, the initialization period is negligibly short compared to the whole training ($\lambda \ll 500$).

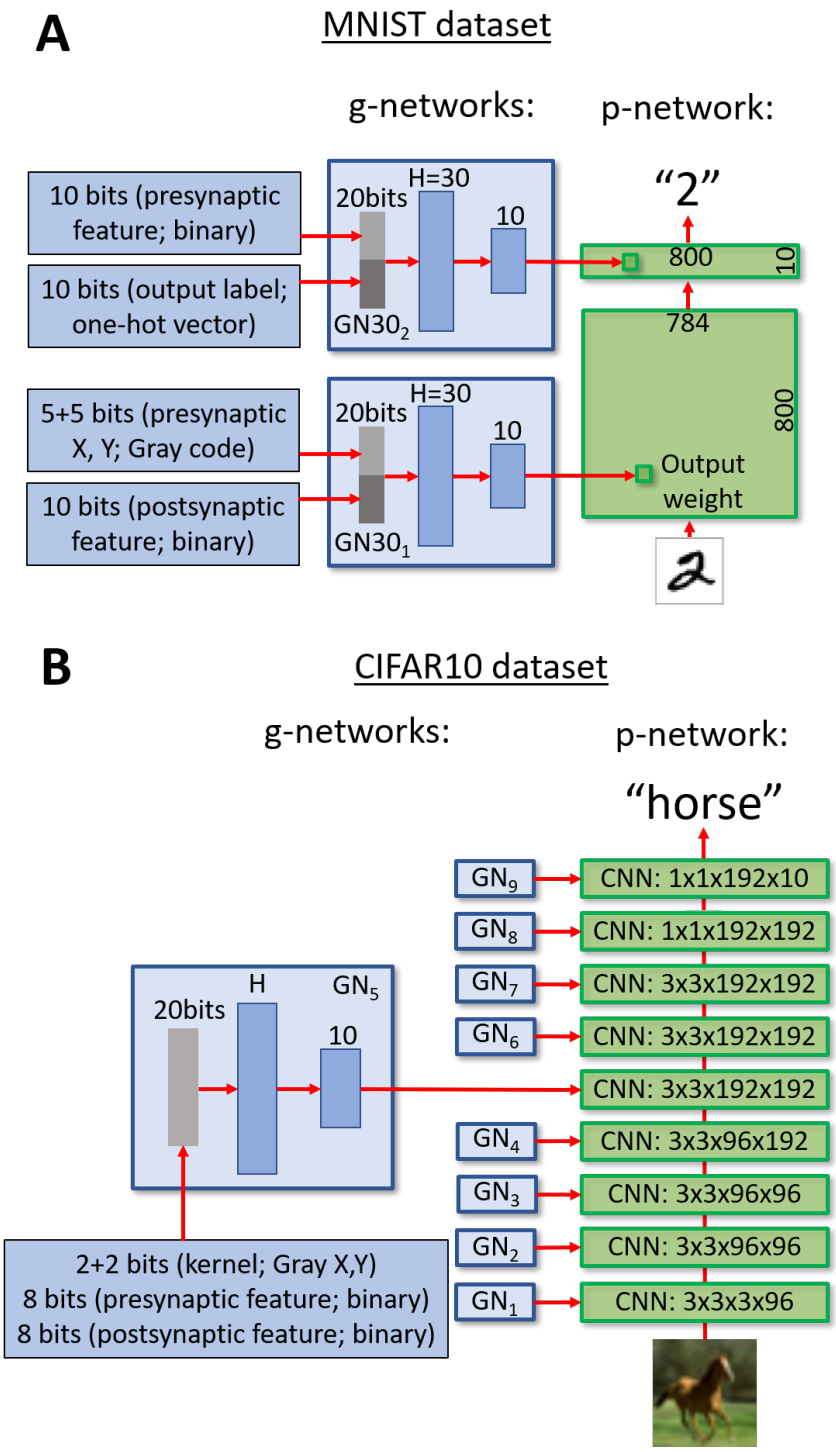

Figure S.1: The structure of g-networks for MNIST (A) and CIFAR10 (B) datasets.

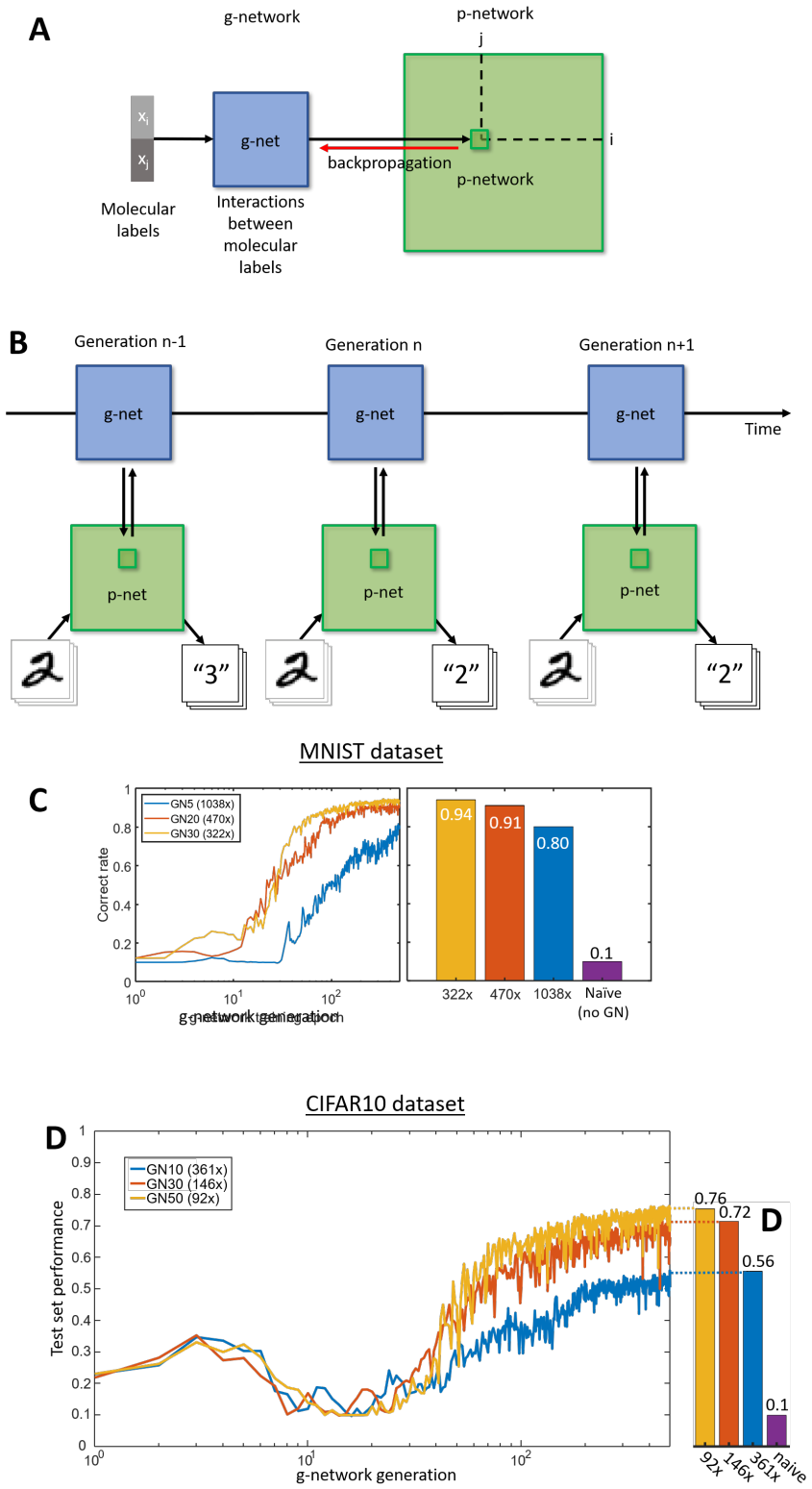

Figure S.2: Training strategies of g-networks. (A) End-to-end backpropagation of errors from the output of the p-network to g-networks. This implementation is slow since it involves generating a complete set of p-network weights and biases for each p-network minibatch. (B) The intermittent training strategy. G-network of generation n-1 is used to generate the p-network (down arrow, generation n). The p-network is trained using several minibatches without backpropagation of the gradients into g-network. Then, the g-network is trained to match the adjusted p-network (up arrow in generation n). The resulting g-network in generation n is used to generate the p-networks in the next step (n+1). (C) The dynamics of training of g-network for the MNIST dataset. (D) The dynamics of training for CIFAR10 dataset. The small bump in performance at generations 2-7 is due to the annealing strategy used to initialize g-networks in this case [Eq. (3)].

References

Baldwin, J. M. (1896). *Physical and social heredity*.

Barabási, D. L. and Barabási, A.-L. (2020). A genetic model of the connectome. *Neuron*, 105(3):435–445.

Bargmann, C. I. (1998). Neurobiology of the caenorhabditis elegans genome. *Science*, 282(5396):2028–2033.

Chklovskii, D. B., Schikorski, T., and Stevens, C. F. (2002). Wiring optimization in cortical circuits. *Neuron*, 34(3):341–347.

Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J. (2020). A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43.

Frank, G. (1953). Pulse code communication. US Patent 2,632,058.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Gaier, A. and Ha, D. (2019). Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, pages 5364–5378.

Goodhill, G. J. and Baier, H. (1998). Axon guidance: stretching gradients to the limit. *Neural computation*, 10(3):521–527.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex systems*, 1(3):495–502.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106.

Itzkovitz, S., Baruch, L., Shapiro, E., and Segal, E. (2008). Geometric constraints on neuronal connectivity facilitate a concise synaptic adhesive code. *Proceedings of the National Academy of Sciences*, 105(27):9278–9283.

Kepecs, A. and Fishell, G. (2014). Interneuron cell types are fit to function. *Nature*, 505(7483):318–326.

Langston, R. F., Ainge, J. A., Couey, J. J., Canto, C. B., Bjerknes, T. L., Witter, M. P., Moser, E. I., and Moser, M.-B. (2010). Development of the spatial representation system in the rat. *Science*, 328(5985):1576–1580.

LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

LeDoux, J. (2012). Rethinking the emotional brain. *Neuron*, 73(4):653–676.

Linsker, R. (1986). From basic network principles to neural architecture: Emergence of spatial-opponent cells. *Proceedings of the National Academy of Sciences*, 83(19):7508–7512.

Rodieck, R. W. and Rodieck, R. W. (1998). *The first steps in seeing*, volume 1. Sinauer Associates Sunderland, MA.

Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *2013 12th International Conference on Document Analysis and Recognition*, volume 3, page 958, Los Alamitos, CA, USA. IEEE Computer Society.

Sperry, R. W. (1963). Chemoaffinity in the orderly growth of nerve fiber patterns and connections. *Proceedings of the National Academy of Sciences of the United States of America*, 50(4):703.

Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Tessier-Lavigne, M. and Goodman, C. S. (1996). The molecular biology of axon guidance. *Science*, 274(5290):1123–1133.

Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057*.

Von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100.

Wei, Y., Tsigankov, D., and Koulakov, A. (2013). The molecular basis for the development of neural maps. *Annals of the New York Academy of Sciences*, 1305(1):44–60.

Zador, A. M. (2019). A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7.

Zipursky, S. L. and Sanes, J. R. (2010). Chemoaffinity revisited: dscams, protocadherins, and neural circuit assembly. *Cell*, 143(3):343–353.