2018

Developing Toward Generality: Combating Catastrophic Forgetting with Developmental Compression

Shawn L. Beaulieu
University of Vermont

Developing Toward Generality: Combating Catastrophic Forgetting with Developmental Compression

A Thesis Presented

by

Shawn L.E. Beaulieu

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Complex Systems and Data Science

May, 2018

Defense Date: March 29th, 2018
Thesis Examination Committee:

Josh C. Bongard, Ph.D., Advisor
Peter Sheridan Dodds, Ph.D., Chairperson
Chris Danforth, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of the Graduate College

Abstract

General intelligence is the exhibition of intelligent behavior across multiple problems in a variety of settings, however intelligence is defined and measured. Endemic in approaches to realize such intelligence in machines is catastrophic forgetting, in which sequential learning corrupts knowledge obtained earlier in the sequence or in which tasks antagonistically compete for system resources. Methods for obviating catastrophic forgetting have either sought to identify and preserve features of the system necessary to solve one problem when learning to solve another, or enforce modularity such that minimally overlapping sub-functions contain task-specific knowledge. While successful in some domains, both approaches scale poorly because they require larger architectures as the number of training instances grows, causing different parts of the system to specialize for separate subsets of the data. Presented here is a method called developmental compression that addresses catastrophic forgetting in the neural networks of embodied agents. It exploits the mild impacts of developmental mutations to lessen adverse changes to previously evolved capabilities and 'compresses' specialized neural networks into a single generalized one. In the absence of domain knowledge, developmental compression produces systems that avoid overt specialization, alleviating the need to engineer a bespoke system for every task permutation, and does so in a way that suggests better scalability than existing approaches. This method is validated on a robot control problem and may be extended to other machine learning domains in the future.

Keywords: neural networks, genetic programming, catastrophic forgetting

Acknowledgements

There are two people without whom this thesis wouldn't exist: Dr. Josh Bongard and Sam Kriegman, who supplied invaluable insight, criticism, and mentorship. I would also like to thank the Morphology, Evolution, and Cognition Laboratory as well as the Vermont Complex Systems Center, and in particular Dr. Peter Sheridan Dodds, for creating an amiable and stimulating work environment. Extreme gratitude must also be expressed for the professional guidance of Dr. Doina Precup and Dr. Josie Dostie of McGill University, as well as Dr. Michael C. Hallett and Dr. Vanessa Dumeaux of Concordia University. I would be remiss if I didn't also acknowledge the kindness and patience of my friends and family. And last I would like to thank the Vermont Advanced Computing Core supported by NASA (NNX 06AC88G), at the University of Vermont for providing the computational resources that made the results reported in this paper possible.

Table of Contents

List of Figures

List of Tables

1

Theoretical Groundwork.

If intelligence isn't *all* that matters in life [1], it certainly constitutes the lion's share of meaning, value, and purpose—or at least is the mechanism by which we obtain these things. Everything about which we should rationally care (health, wealth, knowledge, etc.) can be best cultivated by the practices and institutions that spring from communities of sufficiently intelligent minds [2]. Various attempts have been made to make concrete the gossamer-like notion of intelligence, only to have nature or

reason puncture the narrative. Many of the explanations that have survived cultural and scientific selection are either so baroque as to be unintelligible or too simplistic to be anything but vacuous [3]. However, most appraisals of the mind have converged on *problem solving* as a fundamental feature of intelligence (where definitions for "problem" and "solution" are left regrettably vague). In this light, intelligence is no longer the sole province of brains, but of any set of properly assembled computational units, regardless of the method by which they exchange information. This flavor of universality, called *substrate independence* [4], is the motivating principle behind the project of actualizing machine intelligence.

The advent of deep learning [5] has incited both hysteria and trenchant opposition to its claims and methodologies [6, 7]. But irrespective of the relative (de)merits of deep learning, the field's most enduring contribution to the broader scientific enterprise will be its rebirth of an old technology: neural networks. These powerful function approximators, which will be plumbed more completely in the following sections, are what fuel the (appearance of [8, 9]) intelligent behavior in systems ranging from simple autonomous quadrupeds to Go virtuosos [10].

Although intellectually stimulating, deep learning systems are rarely deployed in the wild, and when they are they're narrowly constrained to a singular task (e.g. object recognition) while the rest of the apparatus runs on more traditional algorithms. The reasons for this are manifold. Among them are the failure of such systems to adequately bridge the reality gap [11] and their inherent susceptibility to catastrophic interference [12, 13], which impairs their ability to integrate new information into existing models. Critically, deep methods are also extremely data hungry, so as to compensate for their independence from perpetual human supervision. For most

problems, human designed solutions, built atop tried and true techniques and modes of analysis, outperform *denovo* models like those deep learning has produced. Deficiencies such as these make for brittle programs that flounder outside their narrow domain of applicability.

Algorithms that can crack these barriers to progress, particularly those that inhibit generalization, are the central focus of this thesis. More concretely, a method for mollifying catastrophic interference called *developmental compression* will be introduced with accompanying speculation on future improvements and applications.

1.1 Neural Networks.

Artificial neural networks (ANNs) are virtual simulacra of the chattering tangle of cells that compose the brains and nervous systems of animals. Originally conceived as explanatory models of neurological phenomena [14], neural networks have since been amended by increasingly many abstractions, and repurposed for statistical and machine learning. One reason for their ubiquity is their theoretical capacity for approximating any continuous function, f, to an arbitrarily low degree of error, $\epsilon > 0$, as shown by the universal approximation theorem [15]. Neural networks are characterized by layers of idealized neurons that compute functions which are communicated to other layers via weights, or synapses. The most basic neural architecture is one in which an input layer, represented by features in a dataset or pixels in an image, feeds into an output layer that calculates some *prediction*, $f(\mathbf{x})$, through an affine transformation—mediated by a single set of *weights*, W, that dampen or amplify signals from the input—followed by a point-wise nonlinear *activation* (e.g. hyperbolic tangent).

$$f(\mathbf{x}) = \tanh(W\mathbf{x} + b) \tag{1.1}$$

Where b is a bias vector (which can be subsumed by the weight matrix provided a corresponding column of ones is appended to the input data).

Summarizing this process, the input first undergoes a linear transformation, then translation, and finally nonlinear transformation. Learning consists in moving from a set of randomly initialized weights to those that allow the network to produce a desirable output, as determined by a *fitness metric* or *loss function* for evolutionary and gradient-based optimization, respectively. Output types can range from categories (e.g. "dog", "cat", etc.), to standard regressions, or even plausible samples from the distribution from which the training data is thought to emerge [16].

Interstitial hidden layers (Figure 1.1) of various dimensions and properties [18, 19], linked together by sheets of adaptable weights, can be added to the network to further transform the data such that its underlying structure can be better deduced [17]. Unbraiding tightly woven features in this way requires that the data we observe assembles into a manifold, or is concentrated near such a space, that is low-dimensional relative to the corresponding embedding space, otherwise known as the *manifold hypothesis* [20]. So-called deep networks are those containing many hidden layers, and can be seen through the lens of statistical learning as maximum likelihood estimation on a recursive generalized linear model [21]. For a more robust treatment of the underlying theory of neural networks, see [17, 20].

Modern architectures, studded with hundreds of layers and billions of parameters [22], are decidedly more ostentatious than the modest blueprints presented here. However, if the goal of science is not merely to obtain good outcomes on highly

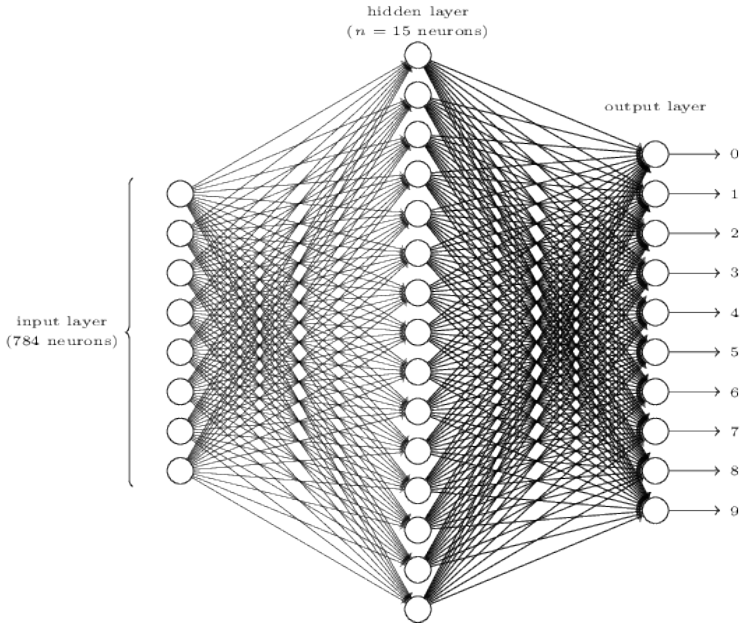


Figure 1.1: *A three layer neural network. One hidden layer containing 15 neurons sandwiched between an input layer of 784 neurons (i.e. pixels) and 10 output neurons (i.e. classes) [17])*

contrived tasks but to produce progressively better *explanations* [23], then such large networks are only obfuscatory. Because they supply such a glut of computational power they render the relatively simple problems for which they're deployed immediately more tractable—as evidenced by random search competing favorably against more fashionable techniques [24]. Indeed, sufficiently vast networks have the representational capacity for superficially impressive behavior, despite containing myriad deficiencies [25]. As problems in science and engineering inevitably become more complex, the strategy of adding more compute quickly becomes infeasible. In order to address systemic defects like catastrophic forgetting, it's necessary to work with simple networks whose properties under modification are transparent to practitioners.

Despite the seemingly large chasm that separates deep neural networks from

comparatively shallow ones, extant strategies for optimizing the parameters of a network ($\theta = \{W, b\}$), regardless of size, fall into two basic categories, each with distinct advantages and motivating philosophies. These are the aforementioned learning methods: evolutionary and gradient-based optimization.

1.2 Gradient Methods.

For a randomly initialized network, the output of a single forward pass will be woefully inaccurate. Training the network to shrink the discrepancy between the actual and desired output is an exercise in becoming less wrong. In supervised learning, the suite of tasks for which gradient methods are best suited [26], there exists some unimpeachable truth against which the network can compare its hypotheses about the training data. Doing so yields a score, whose precise form varies depending on the loss metric used (e.g. cross-entropy, Kullback-Liebler divergence, etc.), that can then be communicated between the free parameters of the network. The ultimate goal of learning is to change these parameters so as to optimize the score obtained. Such change is facilitated by identifying how the score varies as a given parameter varies, which is quantified by the derivative of the score with respect to that parameter. Given that each layer, l_n, depends on the immediately prior layer, l_{n-1}, derivatives for earlier layers in the network can be computed by successive application of the chain rule. This is known as *backpropagation of error*. Below is vectorized backpropagation for the two layer network discussed above.

$$\frac{\partial L(\mathbf{x};\theta)}{\partial \theta} = \frac{\partial L(\mathbf{x};\theta)}{\partial f(\mathbf{z})}\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \theta} = \frac{\partial L(\mathbf{x};\theta)}{\partial f(\mathbf{z})}\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\frac{\partial (W\mathbf{x}+b)}{\partial \theta} \tag{1.2}$$

Once the derivatives for the network are computed, the parameters are updated by an optimization technique called *gradient descent*, in which the value of each parameter is iteratively moved by a finite amount in the direction opposite steepest ascent (i.e. the direction of the gradient, as expressed by the cosine of the angle between a unit vector and the function in question):

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla f_\theta(\mathbf{x}) \tag{1.3}$$

Alternatively, this process can be reformulated as finding the first-order Taylor series approximation to the neural network in the neighborhood of the current hypothesis. The parameters of the network are then modified such that the contribution of each parameter to the global loss function decreases for subsequent iterations.

Assume the direction and magnitude of change are dictated by a vector $\mathbf{s}$. Following the logic in the original formulation, we move in the direction of greatest descent by setting $\mathbf{s}$ equal to $-\alpha \nabla f_\theta(\mathbf{x})$, where α governs the rate of learning:

$$f_\theta(\mathbf{x} + \mathbf{s}) \approx f_\theta(\mathbf{x}) + \nabla f_\theta(\mathbf{x}) * \mathbf{s} = f_\theta(\mathbf{x}) - \alpha \nabla f_\theta(\mathbf{x})^T \nabla f_\theta(\mathbf{x}) \tag{1.4}$$

To take a concrete example, suppose we have a set of images composed entirely of dogs and cats, and we want to build a model that reliably distinguishes between the two animals. If the resolution of our images is 28 pixels by 28 pixels, we have 28*28 or 784 input features, represented by nodes in the input layer, and just 2 outputs (dog=0, cat=1). Now suppose we have one hidden layer with 100 sigmoidal neurons that are fully connected to the input and output layers. To train our network, we first define a loss function over the two classes, which in this case is just the negative

logarithmic logistic loss function:

$$L(\mathbf{x};\theta) = \sum_{i=1}^{N} y_i * (\sigma(f_\theta(\mathbf{x}_i)) - (1 - y_i) * (1 - \sigma(f_\theta(\mathbf{x}_i))) \tag{1.5}$$

Here, $\sigma(\cdot)$ is the sigmoid function, or $\frac{1}{1+e^{-\mathbf{z}}}$, and $f_\theta(\mathbf{x}_i)$ is the hypothesis generated by the neural network for instance i. Typically the weights of the network are initially drawn from a truncated Gaussian function, $w_{u,v} \sim \mathcal{N}(0, 1)$, and shrunk by multiplication with a small coefficient (e.g. $1E^{-3}$) to ensure the first pass through the network is functionally random [20]. For each hypothesis, or batch of hypotheses, we find the derivative of the loss with respect to each parameter using chain rule to backpropagate error through each layer, and then recalibrate the network by pushing individual parameters in their direction of largest descent. After sufficiently many passes through the network—a sequence of conjectures and revisions [27]—we hope to have a model that has internalized some meaningful distinction between dogs and cats based purely on the pixels fed to it. To the extent that this ability exists, the network will be able to correctly classify images not seen during training.

1.3 Evolutionary Algorithms.

Survival of the fittest [28] is a concept that for most people evokes a glorious struggle between enemies pitted in a constant battle of brain and brawn. In reality, fitness merely describes an organism's ability to multiply its genes through the creation of progeny, which *may* have the ancillary benefit of developing better vision, sharper claws, or more powerful minds—so long as these traits help promote gene replication [29, 30]. By contrast, fitness in evolutionary computation can be arbitrarily defined to be

anything from task performance (as in gradient methods) to behavioral novelty [31]. While gradient methods excel at supervised learning, where the degree of error is easily computable, evolutionary algorithms shine when an unambiguous loss signal isn't available. These include tasks where the signal for success is sparse, like a labyrinth containing a single prize.

Mutation, which is the primary driver of change in biological evolution, is also the mechanism by which the parameters of evolutionary programs vary. Whereas the locus of natural selection is the gene, evolutionary algorithms act on the parameters of model being evolved. For neural networks, a simple genetic algorithm works by spawning a population of networks of size P, the parameters for which are drawn from a truncated Gaussian—just like in the initialization step for gradient-based optimization. Each member of the population is then evaluated on the given task (e.g. image classification for disembodied agents, or navigation for embodied agents) and is awarded a fitness score that reflects whatever property one hopes to maximize. It's at this point, *reproduction*, that different classes of evolutionary algorithms begin to diverge. Traditional genetic algorithms preserve the population's fittest individuals for generating offspring in the generation that follows. These offspring differ from their parents at a random site through the injection of noise: the virtual equivalent of a single nucleotide polymorphism. Given a two layer neural network with a single matrix of weights, W, mutation impinges on a single cell of that matrix as follows:

$$i \sim U(0, N) \tag{1.6}$$

$$j \sim U(0, N) \tag{1.7}$$

$$W_{i,j}^{t+1} \sim \mathcal{N}(\mu = W_{i,j}^{t}, \sigma = |W_{i,j}^{t}|) \tag{1.8}$$

Where N is the number of nodes in the layer being mutated. This procedure lasts for G generations, after which the individual with the highest fitness is reported as the most plausible solution. On the other hand, evolutionary strategies (ES) [32–34] dynamically alters the collective mean and variance for each parameter across the whole population. By approaching the problem with a population-based metric, genetic computation can be yoked with its gradient-seeking cousin by virtue of its relation to variational optimization [35]. Rather than directly computing the derivative of the network with respect to a continuous and differentiable loss function, which may not exist for the task we want our network to perform, variational optimization (and by extension, evolutionary strategies) seeks to approximate the derivative of the fitness function using a descendant of the notorious REINFORCE algorithm [36]. The foundational principle of variational optimization is that the minimum of a function is always less than or equal to its expected value.

If we assign a distribution, $p(\mathbf{x}|\theta)$, to every parameter across a population, P, where $\theta = \{\mu_{i,j}, \sigma_{i,j}\}$ for parameter i, j in $\mathcal{W}_{u,v}$ (the set of matrices connecting layers u and v), then for a single parameter:

$$\min_{x} f(x) \leq \mathop{\mathbb{E}}_{p(x|\theta)} [f(x))\,] \tag{1.9}$$

Setting $U(\theta)$ equal to the expectation on the right side of (1.9) and taking the partial derivative with respect to θ:

$$\frac{\partial U(\theta)}{\partial \theta} = \mathop{\mathbb{E}}_{p(x|\theta)} [f(x) \frac{\partial}{\partial \theta} \log(p(x|\theta))\,] \tag{1.10}$$

This is derived by expanding the expectation and cleverly rearranging the derivative of the logarithm of $p(x|\theta)$.

$$\frac{\partial \log(p(x|\theta))}{\partial \theta} = \frac{1}{p(x|\theta)} \frac{\partial}{\partial \theta} p(x|\theta) \tag{1.11}$$

$$\frac{\partial p(x|\theta)}{\partial \theta} = p(x|\theta) \frac{\partial p(x|\theta)}{\partial \theta} \tag{1.12}$$

How $U(\theta)$ changes with respect to θ can now be approximated by extracting S samples from the generating distribution, $p(x|\theta)$, and taking the *mean* value of (1.10) across S after obtaining fitness, $f(x)$. With this derivative in hand for each parameter in our network, gradient descent can be used to incrementally improve performance just as in backpropagation. Moreover, if $p(x|\theta)$ is a distribution from which we can easily sample, like a Gaussian, then (1.10) reduces to (1.13) after taking the derivative of $\mathcal{N}(\mu, \sigma)$ with respect to θ and defining $\epsilon = x - \theta$.

$$\frac{\partial U(\theta)}{\partial \theta} = \frac{1}{\sigma^2} \mathop{\mathbb{E}}_{\epsilon \sim \mathcal{N}(0,1)} [\epsilon f(\epsilon + \theta)] \tag{1.13}$$

In words, the gradient can be approximated by disturbing the value of θ at time t with Gaussian noise drawn from ϵ. For a variety of tasks these methods have proven to be competitive with highly byzantine algorithms like deep (recurrent) Q-networks [37, 38], which seek an optimal control policy, $\pi(\mathbf{a}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$—that is, the best action at time t given the previous action and environmental state—through repeated exposure to trials of punishment and reward.

With enough parallel computing, however, even simple genetic algorithms boast near state of the art performance on tasks that were previously thought to be beyond

the purview of genetic computation [24, 34, 39].

Other genetic algorithms include NEAT [40], which offers a principled approach to genetic recombination, and HyperNEAT [41], an indirect encoding mechanism that extends NEAT to the evolution of weights and nodes in a compositional pattern-producing network that generates neural controllers exhibiting global geometric regularities. Although these algorithms have useful and interesting properties, they are beyond the scope of this thesis. Future references to evolutionary algorithms will pertain to traditional evolutionary techniques.

1.4 Simplicity, Frugality, Transparency.

For all the velocity with which the field of artificial intelligence seems to be moving [10, 16, 42–45] there remain intrinsic flaws that are routinely ignored or brutishly conquered with sheer computational power. Despite numerous attempts to overcome these shortcomings [46–49] none have succeeded in eliminating them altogether. However, each such instance has forged an important step toward a global solution. For the problem of catastrophic interference, prior proposals have tried to foster implicit and explicit modularization [48] of neural networks, or to obtain multi-purpose synapses through the use of information theory [49].

Although seeking modularity is reasonable when the set of desired behaviors is small, any discernibly general agent would require networks whose computational cost massively exceeded their benefits to accommodate all the relevant modules. More recent attempts have sought to train a generalist master network to slowly incorporate into itself task-specific knowledge obtained by a crowd of specialist acolytes [45]. However, given that the master network trains itself on all acolyte data concurrently,

this isn't so much an attempt to overcome catastrophic forgetting as it is an effort in multi-tasking, which is a separate (but related) endeavor.

In the interest of promoting the three cardinal values for which this section is named, the algorithm put forth in this thesis uses a two-layer neural network, whose parameters are optimized by a simple genetic algorithm. Evolution is the preferred method in this case due to the nature of the problem and the costs associated with the alternative [37]—all of which will be discussed in Chapter 3. Additionally, it's typically wise to validate candidate solutions on tasks that are simple before moving on to more challenging frontiers. This is analogous to Ben Recht's Linearization Principle [7], which loosely states that any system whose behavior is wrong on problems that are linear is likely to *notevenbewrong* for nonlinear problems.

2

Catastrophic Forgetting.

Like many who came before them, and countless others who will succeed them, the progenitors of the Enlightenment wrongly conceived of humans as essentially rational actors who behave so as to maximize their well-being, and who can be persuaded of the "right" position if presented with sufficiently compelling evidence. Sadly, decades of research in behavioral and social psychology [50,51] has eroded this faith in the primacy of reason. Humans are prone to a constellation of biases, cognitive fallacies, and

flights of emotion that defy their best interests. Tribal loyalties extinguish our better judgment and spawn resentment and hostility toward groups defined by arbitrary and superficial prejudices. We hold grudges. We're narrowly self-interested. We're parochial. We're also the most intelligent species on the planet. This is all to say that despite the pretense to generality in human intelligence, there exist entrenched defects in our cognition.

Similarly, neural networks, whose sophistication is eclipsed by even the most rudimentary biological organism, suffer flaws that are the product of their design. One such flaw is catastrophic interference—now more commonly known as *catastrophic forgetting*. Forgetting occurs when an agent is asked to perform sequential learning on two or more (sub)sets of data of a larger data structure, $\mathcal{D}$. If the data are fully accessible to the agent during training, the problem of forgetting largely evaporates [45, 52]. However, if changes to the network (by mutation or backpropagation) cause improvements in one task while diminishing or neutralizing performance in another, this can also be construed as catastrophic forgetting.

Intuitively, networks forget in this way because whatever competencies result from a particular parameterization will be degraded when that parameterization is sufficiently altered. As a result, two broad classes of defense against catastrophic forgetting have been proposed: modularization and variable synaptic plasticity.

2.1 Modularization.

If catastrophic forgetting is the consequence of modifications to hard-won knowledge contained in the synapses of neural networks, one solution might be to prevent such deleterious modifications from happening. Modularization [48, 53–55] achieves this by

allocating marginally overlapping fractions of a network's total capacity to specific traits and behaviors. This helps ensure that changes to the network that enhance performance in one task don't interfere with the performance already obtained on another [56–58].

The most recent such manifestation of this approach is called *diffusion-based neuromodulation* [55]. When presented with a particular task, T point sources (corresponding to the T tasks) issue either inhibitory or excitatory signals, which decay in strength like a Gaussian away from their source, in a two-dimensional space in which the network is embedded. Different subsets of the network are arranged in the embedding space so as to receive only one such signal. This results in a network whose subgraphs can be selectively turned on or off by feedback from the environment. Conditioning the network on an environmental cue in this way allows it to compute unique functions under each such cue.

Omitting the highly constrained nature of a technique like diffusion-based neuromodulation, modularity in general is plagued by a problem of scale. If the suite of tasks requires that an agent exhibit scores of diverse traits, the neural controller must expand by a proportional degree to support their instantiation. As problems complexify, modularity alone is insufficient to militate against forgetting. Nevertheless, modularity may be necessary for the creation of general intelligence, insofar as human intelligence is considered "general" [59].

2.2 Variable Synaptic Plasticity.

Guarding against the degradation of previously acquired knowledge is also the driving principle behind a family of algorithms that seek to dynamically alter the plasticity—

that is, the susceptibility to change—of synapses in a network. Neurobiological studies [60, 61] have demonstrated that in response to novel sensorimotor stimuli, mammalian brains generate a profusion of dendritic spines—channels by which innervating axons transmit electrical information. Over time, extraneous dentrites languish and die, while the neurological response they were originally made for becomes concentrated in only a few spines. Under optogenetic ablation [61], these remaining dendrites appear to encode for actions that vanish when they fall silent. The virtual analog of this phenomenon, elastic weight consolidation (EWC) [49], is the latest entry in a catalog of techniques called variable synaptic plasticity.

Rather than enforce modularization, EWC dynamically alters the plasticity of parameters that are likely to be important for one task when training to perform another. How strongly a parameter is fixed is determined by a hyperparamter that weights the importance of one task relative to another. Plausibly, this has the effect of encouraging modularization (in the absence of an explicit objective to do so). However, if the distribution over a given parameter θ_i supports generalizable traits, like the mutual area in a Venn diagram, EWC can hone in on this region and avoid modularization.

By measuring the overlap in Fisher information of the optimal task parameters across all tasks, the authors claim that modularization is largely averted, but fail to apply more traditional measures of modularity.

To illustrate how elastic weight consolidation works, consider a case where a network must learn to represent two datasets, D_A and D_B. If we want to identify the network parameters, θ, that are most probable given the observed data, then by Bayes's theorem:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{2.1}$$

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}) \tag{2.2}$$

$$\log p(\theta|\mathcal{D}) = \log p(D_A|\theta)p(D_B|\theta) + \log p(\theta) - \log p(D_A)p(D_B) \tag{2.3}$$

$$\log p(\theta|\mathcal{D}) = \log p(D_B|\theta) + \log p(D_A|\theta) + \log p(\theta) - \log p(D_A) - \log p(D_B) \tag{2.4}$$

Recognizing that the middle three terms equal $p(\theta|D_A)$:

$$\log p(\theta|\mathcal{D}) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B) \tag{2.5}$$

This maps to the idea that when we train the network on D_B, the parameters that maximize Equation (2.5) depend on those that have been obtained for D_A exclusively through $\log p(\theta|D_A)$. However, the intractability of the posterior distribution in Equation (2.5) forces us to use a more easily computable surrogate. Generalizing to many tasks, the authors of [49] use the empirical Fisher information matrix to approximate the amount of information about θ carried by an arbitrary collection of instances in a given domain D_T. This yields a new loss function, which includes weight consolidation as a regularization penalty, articulated by the sum squared difference between parameters for the current task, T,and those that have been optimized for another task, T'. This discrepancy is weighted by the Fisher information matrix for task T' and ultimately by the importance hyperparameter, λ:

$$\mathcal{L}(\theta) = \mathcal{L}_T(\theta) + \frac{\lambda}{2}\sum_{i}^{\Theta} F_i^{T'}(\theta_i - \theta^*_{T',i})^2 \tag{2.6}$$

Conveniently, empirical Fisher information is calculated as the Monte Carlo ap-

proximation of the expected squared gradient of $\log p(y|x, \theta)$, which is found as a byproduct of backpropagation [62]. Consequently, EWC adds little to the complexity of training.

Although the authors successfully validate their method on a series of simple tasks, there's a lurking assumption that might prevent EWC from working for more complex problems. Equation (2.5) has such a convenient form because of a naive assumption that D_A and D_B are independent. Various other simplifications have also been made to get a grip on the posterior distribution over the network parameters. Reductionist strategies of this sort, which also form the basis of Naive Bayes classifiers, may be computationally tractable but they fail to appreciate the full sprawl of the phenomena they purport to model.

2.3 Generalizing by Compression.

In a similar vein to elastic weight consolidation, evolutionary methods for attenuating synaptic variability include tuning mutation rates [63] and/or crossover events [64]. Safe mutations [65], which veer from the sacralized principle of random mutation, try to ensure that synaptic changes don't significantly deteriorate behavioral proficiency. Imposing limits on mutation, selection, and reproduction—inching ever closer to gradient methods—has been in circulation since at least the creation of semantic variation operators [66–69]. When replacing old genetic material, these methods insert new content with similar semantic value.

Developmental compression is the idea that catastrophic forgetting can be overcome by giving embodied agents the freedom to alter their genetic composition over their lifetimes, so as to dampen the behavioral impact of any individual genetic mutation.

This is in contrast to the single discrete change in genetic makeup that typically occurs in genetic programming. By virtue of the Baldwin effect [70, 71], beneficial late-onset mutations become canalized as evolution proceeds—shortening the developmental time-scale as progress accumulates.

In the sections that follow, after fully articulating the compression algorithm (Section 3.1.5), modest gains against catastrophic forgetting in a simple robot control problem are shown (Section 3.2)

3

Developmental Compression.

3.1 Methods.

Developmental compression was tested on a robot control problem for three reasons. First, the continuous control of legged robots is a notoriously difficult machine learning problem. Second, by dint of the problem's inherent difficulty, if a robot is exposed to multiple disparate environments, catastrophic forgetting is likely to occur. That is,

improvements to the robot's ability in one training environment will likely disrupt previously-evolved capabilities in other environments. Third, a robot's behavior at one time step has a cascading effect on its behavior in future time steps; so, if fitness is integrated over multiple time steps, and developmental change in control policy is allowed, developmental mutations that manifest late in the life of individuals will impact overall behavior less. This is in contrast with non-developmental mutations which, by definition, take effect at the first time step of a robot's evaluation. This difference is exploited by developmental compression, as explained in 3.1.5.

3.1.1 The robot.

The robot and its environment were simulated using Pyrosim[1] [72], an open source robot- and Python wrapper built atop the Open Dynamics Engine (ODE) physics engine [73] (Fig. 3.1).

The architecture, or morphology, of the robot is generic by design. It was not selected to optimally solve the task it was given and, indeed, was chosen before the task environment was fully specified. It is characterized by a rectangular abdomen, attached to which are four legs, each composed of an upper and lower cylindrical object. The knee and the hip joint of each leg contain a rotational hinge joint with one-degree-of-freedom. Each hinge joint can flex inward or extend outward by up to 90 degrees away from its initial angle. The orientations of the hip and knee joints are set such that each leg moves within the plane defined by its upper and lower leg components.

Inside each lower leg is a touch sensor neuron, which at every time step detects

[1] ccappelle.github.io/pyrosim

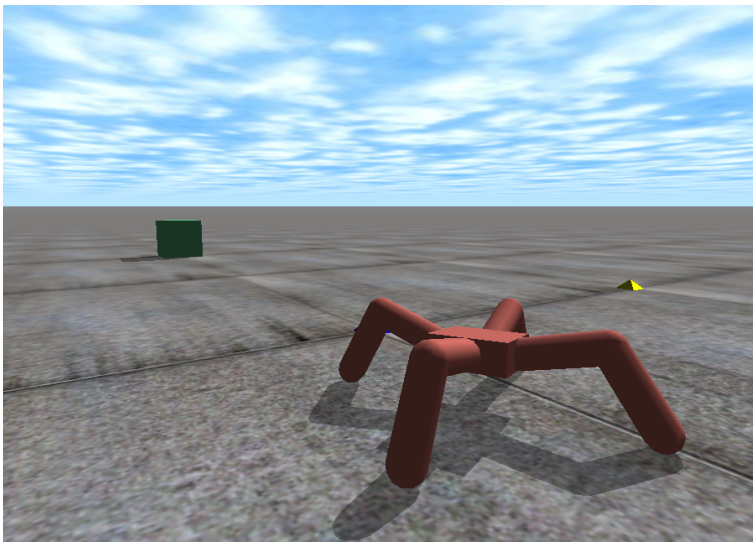


Figure 3.1: *A screen shot of environment A, in which a light-emitting block is placed 30 body lengths in front of the robot's starting position. In environment B, the block is placed 30 body lengths behind the robot. The robot is to perform phototaxis with an on-board sensor that detects light intensity according to the inverse square law.)*

when the lower leg to which it belongs makes contact with the environment. It takes on either -1 (no contact) or $+1$ (contact). Apart from the four touch sensors, a light sensor (whose output is a floating-point number) is embedded in the abdomen of the robot. Due to the inverse square law of light propagation, the light sensor is set to $\ell = 1/d^2$, where d is the Euclidean distance between the light sensor and the light source. Motor neurons innervate each of the eight joints of the robot and enable it to move.

3.1.2 The controller.

In total, there are five sensor neurons fully connected to eight motor neurons such that each sensor neuron feeds into every motor neuron. No hidden neurons were employed.

How the set of sensor neurons connects to, and communicates with, the motor neurons is determined by the robot's neural controller. The edges of the network are represented by a weighted adjacency matrix that is optimized by a direct encoding evolutionary algorithm. Synaptic weights are constrained to $[-1, +1]$.

Motor neurons are updated according to

$$m_i^{(t)} = \tanh \left[m_i^{(t-1)} + \tau_i \sum_{j=1}^{j=5} w_{ji} s_j^{(t)} \right] \tag{3.1}$$

where $m_i^{(t)}$ denotes the value of the ith motor neuron at the current time step, $m_i^{(t-1)}$ is a momentum term that guards against 'jitter' (high-speed and continuous reversals in the angular velocity of a joint), τ_i is a time constant that can strengthen or weaken the influence of sensation on the ith motor neuron relative to its momentum, and w_{ji} is the weight of the synapse connecting the jth sensor neuron to the ith motor neuron. In order to ensure that random controllers produce diverse yet not overly-energetic motion, all τ_i were set 0.3 via empirical investigation.

3.1.3 The task environment.

When the simulation begins the robot is placed at a location that is identically equal to the origin of a smooth two-dimensional plane. Two tasks are presented sequentially to the robot. For the first, a light-emitting box is placed 30 body lengths away from the origin in the *positive* y-direction (that is, *into* the screen); for the second, the box is placed 30 body lengths from the origin in the *negative* y-direction. Success is defined as the ability to walk toward the box in both environments in the alloted time (1000 ts).

3.1.4 The fitness function.

As in the real world, the decay of a light signal's strength is proportional to the inverse squared distance from the source location. Consequently, fitness is taken to be the mean light sensor value experienced by the robot during the simulation. Integrating over time imposes a soft constraint on long peregrinations that obtain good performance only at the end of the evaluation. This selects for behavior of high proficiency throughout the life of an individual. Fitness increases both with proximity to the light source and with increasingly economical behavior that results in quicker travel.

Formally, fitness is defined as:

$$f = \sum_{e=1}^{E} \frac{1}{T} \sum_{t=0}^{T} p_t^{(e)} \tag{3.2}$$

where $p_t^{(e)}$ denotes light sensor's value in environment e at time t

In the experimental treatment each individual is evaluated 4 times, whereas the control treatment has only two evaluations per individual. To ensure equality of resources, the number of generations in the experimental treatment is half the number used in the control treatment.

3.1.5 The compression algorithm.

The intuition for developmental compression is that a given neural controller contains a number of sub-controllers which evolve to specialized, successful behavior in separate training environments. Then, mild developmental mutations gradually 'compress' the specialized sub-controllers into a single base controller, W_0.

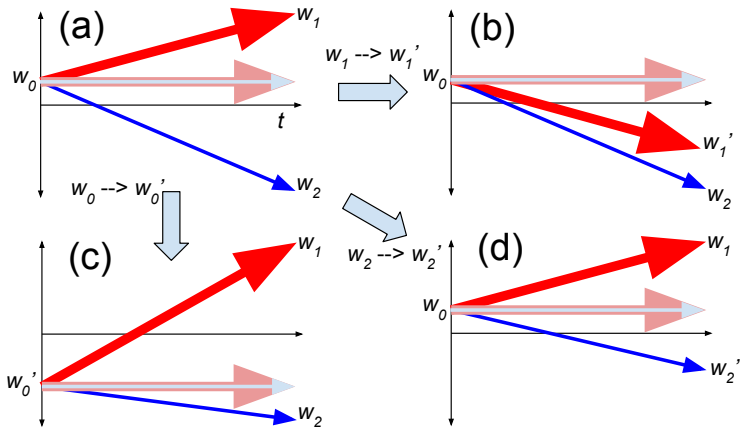


Figure 3.2: *A conceptual visualization of the developmental compression algorithm on a single synaptic weight. (a) Evolution tries to locate a compressed representation of target weights w_1 and w_2. (b) Mutation affects w_1. (c) Mutation affects base w_0. (d) Mutation affects w_2. Horizontal axes represent developmental time.*

More specifically, in two task environments, the networks we seek to compress are those that comprise an individual's genetic tensor ($5 \times 8 \times 3$), represented by three distinct weight matrices (5×8). One such matrix serves as the base controller, while the other two are the target specialist controllers toward which the base linearly develops in the corresponding task environment. Concretely, the zeroth sheet of the tensor, $W_D^{(0)}$, is the base controller while $W_D^{(e)}$ is the target controller for the e^{th} environment.

Development is the process by which compression is enforced. It works by moving the weights of the base state toward a given target state as the simulation proceeds. Under the developmental treatment for the first task, the robot begins the simulation with controller $W_D^{(0)}$ and ends the simulation with controller $W_D^{(1)}$ under a development schedule that's linear in time.

Generically, for a target state belonging to task environment e, every element of

$W_D^{(0)}$ moves toward $W_D^{(e)}$ by:

$$w_{ji}^{(0)} \rightarrow w_{ji}^{(e)} = \alpha_t w_{ji}^{(0)} + (1 - \alpha_t) w_{ji}^{(e)} \tag{3.3}$$

where α is a weight ranging from 1 to 0 that linearly decays with t that moves from 0 to $T - 1$ over T time steps in the evaluation. Thus, the weight of synapse w_{ji} at time t becomes

$$w_{jit} = \frac{(T - 1 - t) w_{ji}^{(0)} + t w_{ji}^{(e)}}{T - 1} \tag{3.4}$$

As the evaluation proceeds, α will dampen the base element and amplify the target element (line 32 in Algorithm 1).

For every task environment, a single agent is evaluated twice: once developmentally and once non-developmentally (lines 23 and 24 in Algorithm 1, respectively). The fitness they obtain in each treatment is then added to their total fitness.

For mutation, a single synaptic weight is chosen at random within the $S \times M \times (e+1)$ weight matrix W: w_{jik}. The new weight of w_{jik} is sampled from a Gaussian distribution whose mean and standard deviation depend on the selected synapse's prior value:

$$w_{jik}^{(\text{new})} \sim \mathcal{N}\left(\mu = w_{jik}^{(\text{old})},\ \sigma = \mid w_{jik}^{(\text{old})} \mid\right) \tag{3.5}$$

If a mutation carries a weight above 1, it is set to 1; if a mutation carries a weight below -1, it is set to -1 (Fig.3.2).

3.1.6 The control algorithm.

The control algorithm optimizes performance by summing fitness scores of non-developmental agents across all task environments. That is, individuals are evaluated once in each environment and the fitness they obtain is added to their respective total fitness, F_p, which is reported once all environments in E have been encountered.

Mutation here mirrors Equation 3.5, except that one synaptic weight is chosen at random from the 5×8 weight matrix, not the genetic tensor.

3.1.7 Random search algorithm.

As a sanity check, the compression and control algorithms were compared with a random search algorithm, in which heredity is discarded. In other words, random search is the control treatment with reproduction removed. However, the ways in which individuals are mutated and evaluated are unchanged.

The total number of robot evaluations across all three algorithms is equalized to ensure a fair comparison.

3.2 Results

We performed 100 independent runs of the experimental treatment (750 generations per run, Sect. 3.1.5), the control treatment (1500 generations, Sect. 3.1.6), and random search (1500 generations, Sect. 3.1.7). Each run was conducted for 1500 generations. At the end of evolutionary optimization, we extracted the fittest individual: the overall run champion, as defined by Equation 3.2. We extracted these champions from each of the runs.

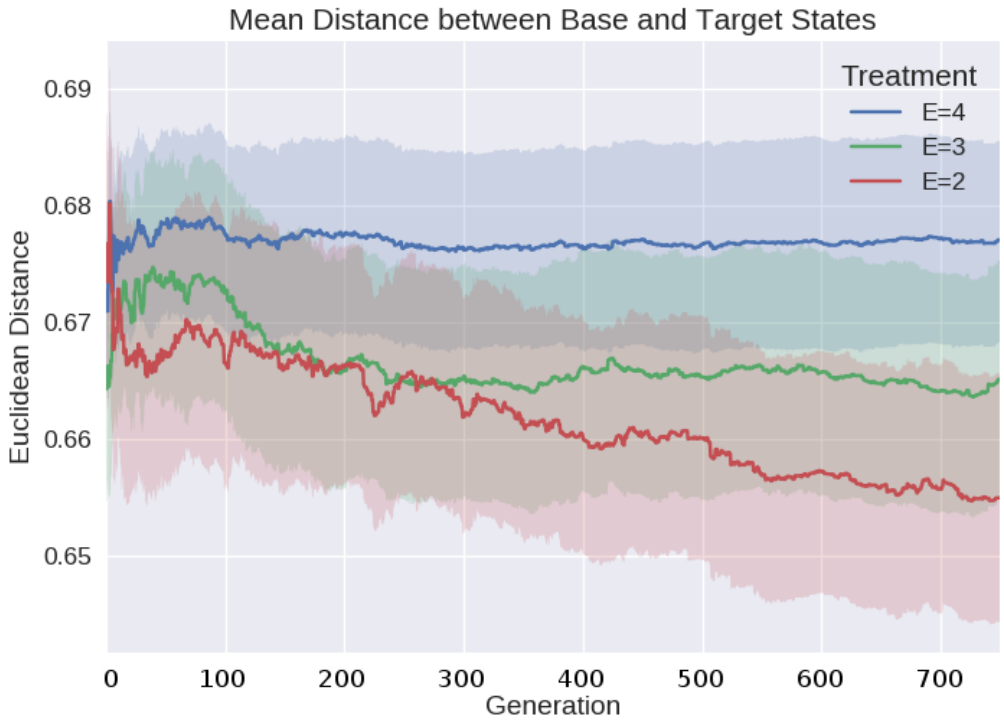


Figure 3.3: *Compression over time. Mean distance during evolution between the base state and target states in the corresponding environments. Values computed using generational run champions across all runs for each treatment (E=2:4) with 95 percent confidence intervals.*

The results presented in Figures 3.5 and 3.4 (generational champions) and 3.1 (overall champions) strictly pertain to non-developmental performance. In the DC treatment, run champions are those whose fitness is highest across all $2E$ evaluations, but we report only the non-developmental performance of those run champions (line 24 in Algorithm 1) for fairness of comparison.

Fig. 3.4a reports the median fitness for the generational champions' best environment for all three treatments. This figure suggests that the control treatment is significantly outperforming both the experimental treatment and random search.

Table 3.1: *Median minimum fitness of run champions.*

	Random	Control	Compression
E=2	0.0826 ± 0.0186	0.0781 ± 0.0372	**0.1103** ± 0.0240
E=3	0.0870 ± 0.0188	0.0847 ± 0.0229	**0.1094** ± 0.0188
E=4	0.0823 ± 0.0142	0.0704 ± 0.023	**0.1016** ± 0.014

However, Fig. 3.5a demonstrates that this conclusion is premature. This figure shows the median fitness of generational run champions in their worst environment. Catastrophic interference is responsible for the discrepancy between the quantity being optimized by the evolutionary algorithm (mean fitness) and the quantity we implicitly seek to optimize (general proficiency).

The ability of robots in the experimental treatment to continue improving in their worst environment is reported in Table 3.1. For the case of two environments ($E = 2$), we assessed statistical significance for the median minimum fitness of each treatment's overall run champion with the Mann-Whitney U test and Bonferroni correction across three possible comparisons (compression vs. control, compression vs. random, random vs. control). We found that developmental compression significantly out performed both the control ($P < 0.0001$) and random treatments ($P < 0.0001$). Results for the median minimum fitnesses of overall run champions are shown in table 3.1.

Minimum fitness is an important metric in this context because if it decreases over evolutionary time it reveals the degree to which an individual narrowly specializes to one task environment. This trend can also arise in the developmental treatment if the base genome evolves to be overly dependent on their scaffolds, sacrificing performance without them. A similar problem can afflict the non-developmental treatments in increasingly many environments, where a good strategy is to avoid costs to overall fitness by doing the same thing in every environment: stand still.

The extent to which development compresses the weight matrices in an agent's controller is shown in 3.3. For every environment, the mean Euclidean distance between the base state and the target state for the generational run champions is computed over the course of evolution. The mean distance is then averaged over the number of environments and reported with a 95 percent confidence interval. At a glance, compressibility appears to correlate with the median minimum fitness plots in Figure 3.5, which suggests that DC's ability to avoid catastrophic forgetting depends on the quality of compression.

By observing where and how individuals fail to achieve optimal performance where they otherwise should, we can glimpse into the logic under which they operate. For example, in the control, a high maximum fitness (Fig.3.4) does not seem to correlate with the ability of phototaxis (Fig.3.5), but rather with the ability to walk remarkably well in one direction. These results suggest that developmental compression was partly able to overcome the large local optima of overspecialization and inaction, while the control and random treatments were not.

Algorithm 1 Developmental Compression Algorithm

$E \leftarrow$ The Task Environments ($1 \times$ number of envs)
$W_S \leftarrow$ Static Genome ($5 \times 8 \times 1$)
$W_D \leftarrow$ Developmental Genome ($5 \times 8 \times$ number of envs)
$G \leftarrow$ Number of Generations

if controlTreatment **then**
    for g=1:G **do**
        CONTROL (W_S, E)
if experimentalTreatment **then**
    for g=1:$G/2$ **do**
        DEVELOPMENTAL_COMPRESSION (W_D, E)

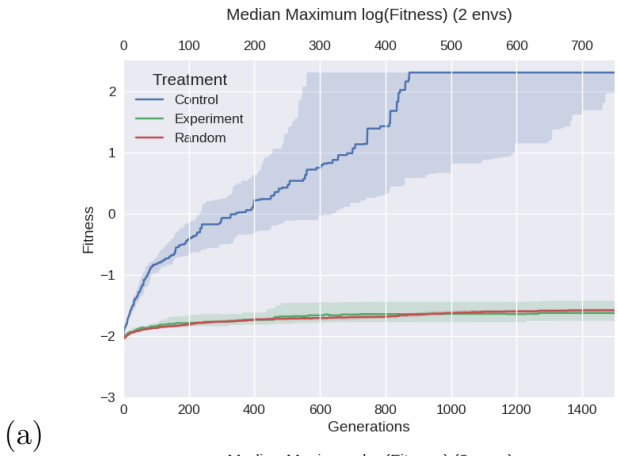
procedure CONTROL(W_S , E)
    for each individual in the population **do**
        `Fitness` = 0
        for e **in** E **do**
            `Fitness` += SIM (e, base = W_S, target = W_S)

procedure DEVELOPMENTAL_COMPRESSION(W_D , E)
    for each individual in the population **do**
        `Fitness` = 0
        for e **in** E **do**
            `Fitness` += SIM (e, base = $W_D^{(0)}$, target = $W_D^{(e)}$)
            `Fitness` += SIM (e, base = $W_D^{(0)}$, target = $W_D^{(0)}$)

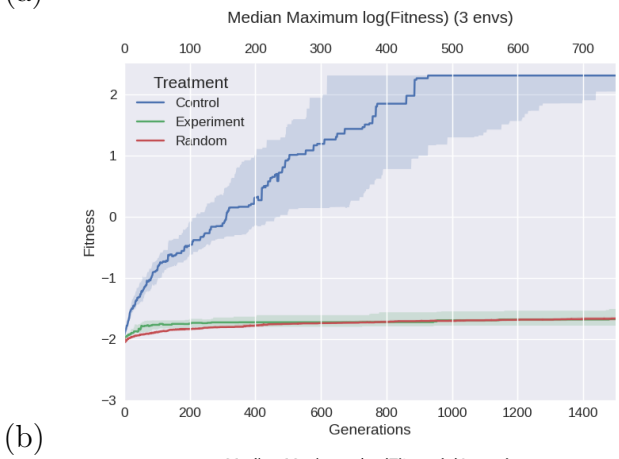
procedure SIM(e, base, target)
    $p_e = 0$ ▷ performance (light intensity)
    for t=1:T **do** ▷ time steps
        for s=1:S **do** ▷ sensor neurons
            for m=1:M **do** ▷ motor neurons

$$w_{smt} = ((T-1-t)w_{sm}^{(\text{base})} + \ tw_{sm}^{(\text{target})})/(T-1)$$

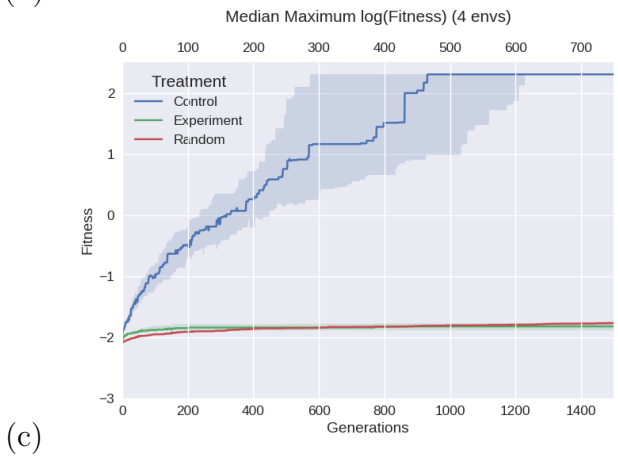
        Perform in env e for time step t using W_t.
        $p^{(e)}$ += $p_t^{(e)}$
    return $p_t^{(e)}$



(a)

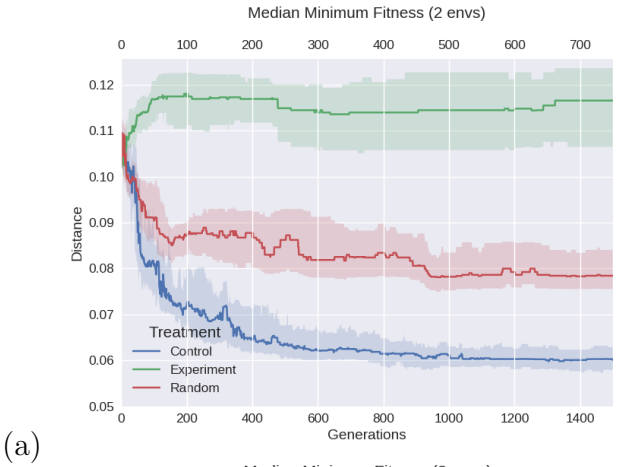


(b)

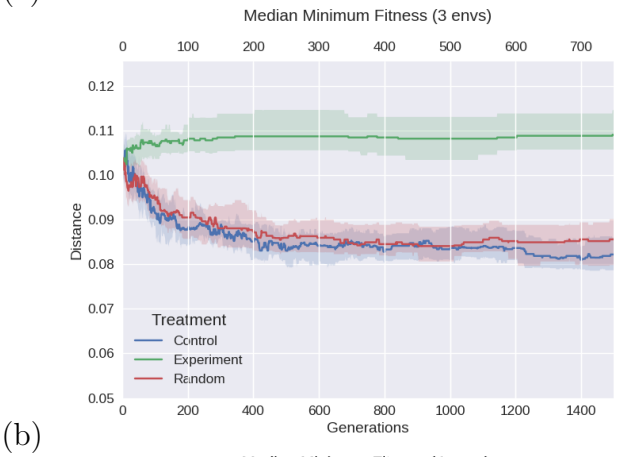


(c)

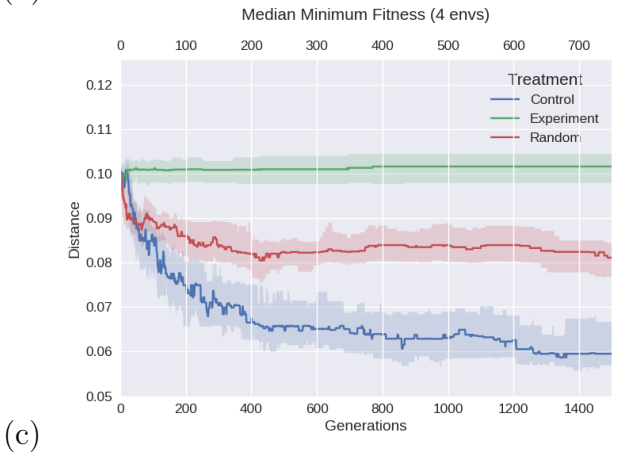
Figure 3.4: *Median Maximum Fitness. The median score across all runs for the maximum fitness value of generational run champions (95 percent confidence intervals). (A) two task environments (E=2); (B) three task environments (E=3); (C) four task environments (E=4). Y-axis is log-scale. For reference, with two environments the experimental treatment has median=0.19, std=0.202.*



(a)



(b)



(c)

Figure 3.5: *Median Minimum Fitness. The median score across all runs for the minimum fitness value of generational run champions (95 percent confidence intervals). (A) two task environments (E=2); (B) three task environments (E=3); (C) four task environments (E=4). This figure highlights catastrophic forgetting in the random and control methods. Control is more adversely affected by forgetting due to overspecialization. The experimental method avoids the negative impact of task antagonism.*

4

Summary and Conclusions

4.1 Discussion

Developmental compression appears to limit specialization, which afflicts both random search and the naive approach of averaging non-developmental performance across all environments (Fig. 3.5). These latter two treatments likely specialize because they suffer from catastrophic forgetting: seemingly redundant improvements in one's best

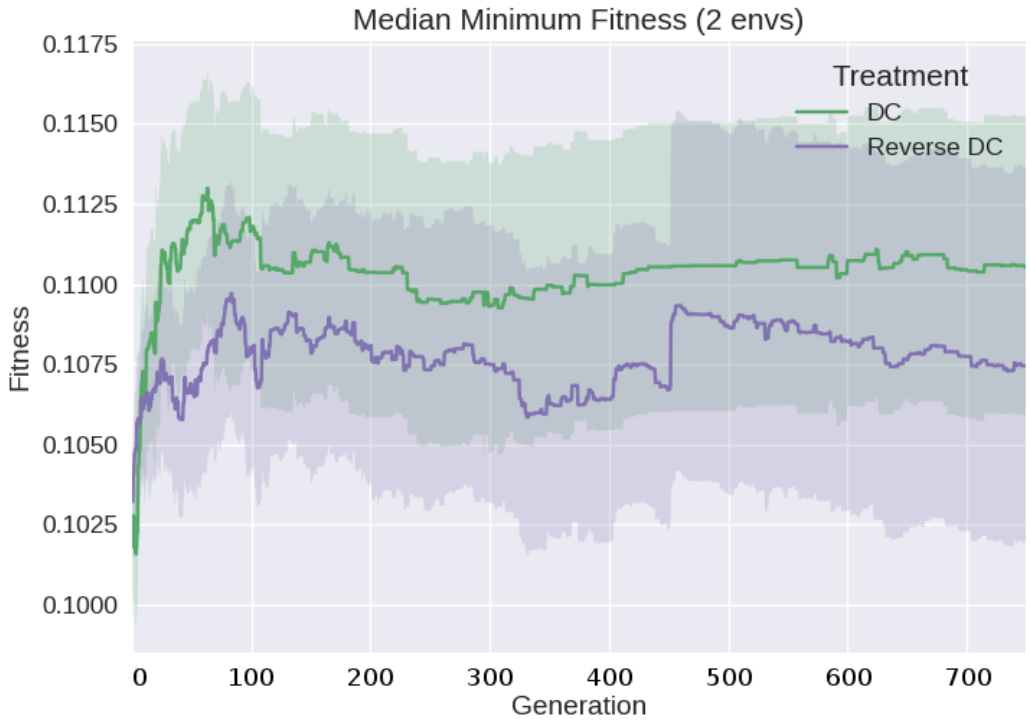


Figure 4.1: *DC vs reverse DC. Comparing the experimental treatment (green) and a reversed version (purple) with 95 percent confidence intervals. See Discussion for details. In 3 and 4 environments reverse DC fails to keep up with DC (results not shown).*

environment recur because such improvements outweigh the penalty to overall fitness incurred by improving one's worst environments. Only robots within the developmental compression paradigm exhibit continued improvement in all environments over evolutionary time (Figs. 3.4a and 3.5a). This is can be ascribed to (at least) two factors.

First, mutations within developmental compression tend to have more mild behavioral effects. This is illustrated in Fig. 3.2. A mutation that strikes a feature in the base matrix, w_0 (Fig. 3.2c) impinges on behavior in all E environments, but this

effect is attenuated as the base matrix develops toward its scaffolding. A mutation that strikes a target matrix (Fig. 3.2b) only modulates behavior in one of the E environments. This is likely to increase the likelihood that the controller may improve in one environment without adversely impacting its performance in the other $E-1$ environments.

The second reason that developmentally compressed controllers outperform those from the random and control treatments at local task generalization is that evolution gradually ignores target scaffolding over evolutionary time. This can be seen in Fig. 3.3, where the distance between the base and target matrices decreases with time. This means that the three matrices within a controller are gradually approaching one another, and the path traversed by development becomes shorter. This is canalization.

The success of developmental compression in this domain is even more surprising given that its fitness landscape is twice as large as the control treatment. However, work on extra-dimensional bypasses [74] has shown that at high dimensions, if an optimization procedure can smooth the fitness space by bridging local optima in lower dimensional spaces, evolvability can be increased. Thus developmental compression can be viewed as a way to safely introduce extra-dimensional bypasses, which may militate against catastrophic forgetting.

4.1.1 Scalability.

Despite its success in two environments, developmental compression weakened in its ability to obtain generalists as the problem was scaled up to three and four environments (Figs. 3.4b,c and 3.5b,c). Indeed, as Fig. 3.3 illustrates, DC also fails to compress increasingly many weight matrices as more problems are introduced.

At present it is difficult to say why this occurs. However, the most likely culprit is a weak evolutionary algorithm and sparse neural controller. An extremely simple genetic algorithm was employed here—one that doesn't attend to diversity, or other salient factors, and thus fails to obtain good and compressible scaffolds on complex problems. In future work we plan to investigate how DC can be strengthened with better evolutionary constraints, and be made to synergize with related algorithms.

Interestingly, the behavioral impacts of mutations become increasingly mild in DC as the number of task environments increases. For two environments ($E = 2$), a mutation to a given target matrix impacts behavior in only one of environment; for three environments ($E = 3$), a mutation to a target matrix impacts behavior in only one of three environments; and so on, with mutation becoming increasingly diluted. While DC might dull the behavioral consequences of any single mutation, there's no guarantee that this will result in good generalists. Instead, this may slow the pace of progress. In future work we will investigate how to address this by using adaptive mutation rates, rather than one that is fixed.

4.1.2 Different forms of development.

In this work, a simple process of development was employed: robots begin with a single controller and move toward E controllers in the corresponding environments. As development proceeds, agents exhibit increasingly unique behaviors. However, different developmental trajectories—ones that aren't strictly linear—might be even better at avoiding catastrophic forgetting.

In this vein, we conducted another set of 100 independent runs of developmental compression in each of two, three, and four environments. But in this case devel-

opmental interpolation was reversed: for developmental evaluations, E base states developed linearly toward a *single* target state (line 23 of Algorithm 1). Curiously, this reversed variation did not appear to avoid catastrophic forgetting as well as the original experimental method (Fig. 4.1). The suggests that the order of compression is important for success in this domain. How compression varies with different developmental programs will be the subject of future investigations. In Fig. 4.1 we only report performance for two environments, but the gap between DC and reverse DC widened more strikingly in three and four environments.

Another potential avenue for improvement is to change the entire process by which development occurs. In its current form, development is uniformly distributed across network parameters and is linear in time. But this need not be true. One could include time as an input to compositional pattern producing networks [41] which would 'paint' different, possibly nonlinear, developmental trajectories onto different synapses. One could also envision employing NEAT [40] to compress base and target neural controllers with different cognitive architectures into a single, non-developing yet generalist architecture. Indeed, prior work has sought to marry network complexification and deep learning [44], which may also be applicable to developmental compression.

4.1.3 Synergies with other methods.

There exist myriad ways to combat catastrophic forgetting, each motivated by specific problems and the obstacles inherent to them. We do not purport to show that developmental compression succeeds where prior methods fail, or that it is preferable in situations where catastrophic forgetting has already been reasonably tempered

[49]. Rather, we believe the results reported in this paper suggest developmental compression may be a potential mechanism for overcoming catastrophic forgetting—either independently or in concert with existing methods.

Although other techniques could be used to resist specialization, they often incur additional costs. For example, an objective that rewards generalists could be included in a multi-objective evolutionary algorithm. However, doing so would increase the dimensionality of the Pareto front, which is known to weaken optimization and requires bulwarking countermeasures [75], thus increasing algorithmic complexity. Another approach would be to formulate a better fitness function. However, this would put us on a frictionless path toward fitness function engineering, which may be appropriate for narrow problems but is likely to produce systems that break outside of their training regime. One could also take the minimum fitness across E to guard against specialization, but this will collapse gradients in the fitness landscape, because mutations that do not affect the worst fitness component aren't seen by search.

Unlike existing methods for combating catastrophic forgetting, developmental compression is expressly designed to minimize the need for supervised intervention in the form of domain knowledge or explicit constraints on behavior. Rather than decide what features of the system are necessary for generalist capabilities, developmental compression lets evolution discover the best way to preserve knowledge through a development program that integrates information obtained over the life of the agent. Given the generality of this method, it is likely to synergize well with existing techniques.

Future implementations will seek to improve developmental compression and to reconcile the results reported here with the work being done in deep neuroevolu-

tion [65]. The Atari-57 game suite and DMLab-30 environments are problems for which developmental compression could be scaled up, modified, and rigorously tested. Whether DC can be parallelized and used to compress populations of specialists across dissimilar tasks into a single master network remains to be seen. If such an ability can be realized, it would represent an evolutionary alternative to the multi-task learning paradigm currently being explored in reinforcement learning [45].

4.2 Concluding Remarks

Developmental compression is a novel technique for avoiding catastrophic forgetting. It works by relying on the smoothing effect of development, and how specialists can thus be gradually and developmentally 'compressed' into a generalist by canalization. It could also work in domains beyond robotics provided two conditions are met. First, each training instance must be extended in time. That is, several forward passes are required to complete one evaluation. Such a condition is usually met by reinforcement learning tasks. Second, performance must be integrated over all time steps of the evaluation. In future work we plan to study the efficacy of developmental compression in machine learning domains outside robotics and in tandem with other advances in the artificial intelligence literature.

Bibliography

[1] Stuart Ritchie. *Intelligence: All That Matters*. Teach Yourself, 2016.

[2] Steven Pinker. *Enlightenment Now: The Case for Reason, Science, Humanism, and Progress*. Penguin Books Limited, 2018.

[3] Shane Legg and Marcus Hutter. A collection of definitions of intelligence. *Self Published*, 2006.

[4] Alan Turing. On computable numbers, with an application to the entscheidungs problem. *The Graduate College at Princeton University*, 1937.

[5] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[6] Gary Marcus. Deep learning: A critical appraisal. *arXiv Preprint. arXiv:1801.00631*, 2018.

[7] Benjamen Recht. An outsider's tour of reinforcement learning. *argmin.net*, 2018.

[8] Justin Glimer, Luke Metz, Faghri Fartash, Samuel S. Schoenholz, Maithra Schoenholz, Martin Wattenberg, and Ian Goodfellow. Adversarial spheres. *arXiv Preprint. arXiv:1801.02774*, 2018.

[9] Alex Irpan. Blog: Deep reinforcement learning doesn't work yet. *alexirpan.com*, 2018.

[10] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv Preprint. arXiv:1712.01815*, 2017.

[11] Jean-Baptiste Moure, Sylvain Koos, and Stéphane Doncieux. Crossing the reality gap: a short introduction to the transferability approach. *Proceedings of the first workshop on the Evolution of Physical Systems (EPS)*, 2012.

[12] Ian Goodfellow J., Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv Preprint. arXiv:1312.6211*, 2013.

[13] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[14] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages Vol.65,No.6, 1958.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, pages 551–500, 1989.

[16] Durk P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv Preprint. arxiv:1312.6114*, 2014.

[17] Michael Nielsen. *Neural Networks and Deep Learning*. 2015.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS'12 Proceedings of the 25th International Conference on Neural Information*, pages 1097–1105, 2012.

[19] Lipton C. Zachary and John Berkowitz. A critical review of recurrent neural networks for sequence learning. *arXiv Preprint. arXiv:1506.00019*, 2015.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[21] Mohamed Shakir. A statistical view of deep learning: Recursive generalized linear models. *The Spectator: A Machine Learning Blog*, 2015.

[22] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv Preprint. arXiv:1701.06538*, 2017.

[23] David Deutsch. *The Beginning of Infinity: Explanations That Transform the World*. New Viking, 2011.

[24] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017.

[25] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv Preprint. arXiv:1611.03530*, 2016.

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv Preprint. arXiv:1412.6806*, 2014.

[27] Karl Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge, 1963.

[28] Charles Darwin and Alfred Wallace. On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection. *Journal of the proceedings of the Linnean Society of London. Zoology*, 3(9):45–62, 1858.

[29] Richard Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

[30] Eli C. Minkoff. *Evolutionary Biology*. Addison-Wesley, 1983.

[31] Joel Lehman and Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life XI*, 2008.

[32] Frank Kursawe. A variant of evolution strategies for vector optimization. *Parallel Problem Solving from Nature*, pages 193–197, 1990.

[33] Oswin Krause, Dídac R. Arbonès, and Christian Igel. Cma-es with optimal covariance update and storage complexity. *30th Conference on Neural Information Processing Systems*, 2016.

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

[35] Joe Staines and David Barber. Variational optimization. *arXiv Preprint. arXiv:1212.4507*, 2012.

[36] Ronald Williams J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, pages 229–256, 1992.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv Preprint. arXiv:1312.5602*, 2013.

[38] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv Preprint. arXiv:1507.06527*, 2015.

[39] Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *arXiv Preprint. arXiv:1712.04170v1*, 2017.

[40] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[41] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[42] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv Preprint. arXiv:1409.0473*, 2014.

[43] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv Preprint. arXiv:1406.2661*, 2014.

[44] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[45] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[46] Sang-Woo Lee, Jin-Hwa Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *arXiv preprint arXiv:1703.08475*, 2017.

[47] Michael E Hasselmo. Avoiding catastrophic forgetting. *Trends in Cognitive Sciences*, 21(6):407–408, 2017.

[48] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. Neural modularity reduces catastrophic forgetting. *The Evolution of Learning: Balancing Adaptivity and Stability in Artificial Agents*, page 111, 2014.

[49] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.

[50] Daniel Kahneman. Maps of bounded rationality: Psychology for behavioral economics.

[51] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica Vol.47 No.2*, pages 263–291, 1979.

[52] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv Preprint. arXiv:1511.06342*, 2016.

[53] Hod Lipson, Jordan B Pollack, Nam P Suh, and P Wainwright. On the origin of modular variation. *Evolution*, 56(8):1549–1556, 2002.

[54] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. In *Proc. R. Soc. B*, volume 280, page 20122863. The Royal Society, 2013.

[55] Roby Velez and Jeff Clune. Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. *arXiv preprint arXiv:1705.07241*, 2017.

[56] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[57] Sang-Woo Lee, Chung-Yeon Lee, Dong-Hyun Kwak, Jiwon Kim, Jeonghee Kim, and Byoung-Tak Zhang. Dual-memory deep learning architectures for lifelong learning of everyday human behaviors. In *IJCAI*, pages 1669–1675, 2016.

[58] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[59] Judith M. Burkart, Michèle N. Schubiger, and Carel P. van Schaik. The evolution of general intelligence. *Behavioral and Brain Sciences, Vol. 40*, 2017.

[60] Guang Yang, Feng Pan, and Wen-Biao Gan. Stably maintained dendritic spines are associated with lifelong memories. *Nature, 462(7275)*, page 920–924, 2009.

[61] Akiko Hayashi-Takagi, Sho Yagishita, Mayumi Nakamura, Fukutoshi Shirai, Yi Wu, Amanda L. Loshbaugh, Brian Kuhlman, Klaus M. Hahn, and Haruo Kasai1. Labeling and optical erasure of synaptic memory traces in the motor cortex. *Nature, 525(7569).*, page 333–338, 2015.

[62] James Martens. New insights and perspectives on the natural gradient method. *arXiv Preprint. arXiv:1412.1193*, 2014.

[63] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *International Conference on Parallel Problem Solving from Nature*, pages 803–813. Springer, 2016.

[64] Jason Teo, Asni Tahir, Norhayati Daut, Nordaliela Mohd Rusli, and Norazlina Khamis. Fixed vs. self-adaptive crossover-first differential evolution. *Applied Mathematical Sciences*, 10(32):1603–1610, 2016.

[65] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O Stanley. Safe mutations for deep and recurrent neural networks through output gradients. *arXiv preprint arXiv:1712.06563*, 2017.

[66] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. A Survey of Semantic Methods in Genetic Programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, 2014.

[67] M. Castelli, L. Vanneschi, and S. Silva. Semantic Search-Based Genetic Programming and the Effect of Intron Deletion. *IEEE Transactions on Cybernetics*, 44(1):103–113, 2014.

[68] Tomasz P Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(3):326–340, 2015.

[69] Marcin Szubert, Anuradha Kodali, Sangram Ganguly, Kamalika Das, and Josh C Bongard. Semantic forward propagation for symbolic regression. In *International Conference on Parallel Problem Solving from Nature*, pages 364–374. Springer, 2016.

[70] George Gaylord Simpson. The baldwin effect. *Evolution, Volume 7, Issue 2*, page 110–117, 1953.

[71] Sam Kriegman, Nick Cheney, and Josh Bongard. How morphological development can guide evolution. *arXiv preprint arXiv:1711.07387*, 2017.

[72] Sam Kriegman, Collin Cappelle, Francesco Corucci, Anton Bernatskiy, Nick Cheney, and Josh C Bongard. Simulating the evolution of soft and rigid-body robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1117–1120. ACM, 2017.

[73] Russell Smith. Open dynamics engine v0.5 user guide. *URL http://www. ode. org/ode-docs. html*, 2008.

[74] Michael Conrad. The geometry of evolution. *BioSystems*, 24(1):61–81, 1990.

[75] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014.