Supplemental Materials for: Liu, Y., Reichle, E. D., & Gao, D.-G. (2012). Using reinforcement learning to examine dynamic attention allocation during reading. *Cognitive Science*. Manuscript submitted for review.

Yanping Liu

Sun Yat-Sen University, China

Erik D. Reichle

University of Pittsburgh, U.S.A.

&

Ding-Guo Gao

Sun Yat-Sen University, China

Note: This is a draft; please do not cite without permission.

Address correspondence to:
Erik D. Reichle
University of Pittsburgh
635 LRDC, 3939 O'Hara St.
Pittsburgh, PA 15260 U.S.A.
e-mail: reichle@pitt.edu

These supplementary materials describe a theoretical framework that specifies how complex behaviors that seemingly require an explicit "teaching signal" for error-driven learning (Rumelhart, Hinton, & Williams, 1986) might instead be acquired via simple reinforcement (Sutton & Barto, 1998). This framework includes algorithms that specify the evolution of artificial neural network topologies capable of learning of large-scale, complex problems using only information about the quality of a network's performance.

Figure 1 is a schematic diagram illustrating our algorithms for implementing macro- and microscopic evolution to generate artificial network topologies that are capable of solving large, complex problems via error-driven reinforcement learning. This process starts by first generating a population of simple genomes that express themselves as individual networks (i.e., as phenotypes). Each network is then trained on the same problem using the (residual) reinforcement-learning algorithm to adjust the network's connection weights. A microscopic evolutionary algorithm (i.e., *CMA-ES*; Hansen, 2006; Hansen & Kern, 2004; Hansen, Müller, & Koumoutsakos, 2003; Hansen & Ostermeier, 2001; Igel, Hansen, & Roth, 2007; Suttorp, Hansen, & Igel, 2009) is then used to "nudge" the evolutionary process towards a better solution. Each network's performance (i.e., fitness) is then evaluated by summing the reward that it received. If the overall fitness of the population fails to improve (i.e., stagnates), then the evolutionary process is "nudged" towards a better solution by increasing the mutation rate via simulated annealing. Finally, the individual networks are allowed to generate the next generation via cloning the fittest individual, and via mutation and crossover. We will now describe this approach in the following three sections, explaining how our approach: (1) selects the appropriate macrostructure (i.e., network topology) and (2) simulates the emergence of the microstructure (i.e., specific connection weights) that support (3) reinforcement learning.

Figure 1

Macroscopic Evolution. Efficient biological evolution entails safeguards to

protect any innovations of the phenotype that offer a selective advantage (in terms of ecological fitness) to the genotype, while simultaneously detecting homology between genotypes and minimizing the structural complexity of the genotype. These safeguards maximize the efficiency of biological evolution by minimizing the overall size of the evolutionary "search space" that must be "traversed" to produce organisms that are fit enough to compete and survive in specific ecological niches. Our method of generating network topologies uses an algorithm (*NeuroEvolution of Augmenting Topologies*, or *NEAT*; Stanley & Miikkulainen, 2002) that was specifically designed to instantiate these three efficiency strategies as follows: First, innovations in network topology are protected (so that they have a reasonable chance of propagating from one generation of networks to the next) by *speciation*, or the evolution of separate species that comprise distinct populations of networks that have unique topologies and that only reproduce within their own population. Second, homologous genotypes can be identified by historical "markers" (i.e., identifiers that reflect each network's evolutionary history) to allow for efficient identification and alignment of the genomes during the crossover stage of reproduction. And finally, the structural complexity of the genotypes is minimized by beginning the process of evolution with the simplest possible network topology—single-layer networks containing only input and output units.

In our approach, each genome is a linear array of genes that represents each network's topology. There are two basic types of genes: (1) *node genes* that determine the functional role of each node (i.e., input, output, or hidden unit) in the network, and (2) *connection genes* that determine the patterns of connectivity among the nodes in the network. Figure 2 is a schematic diagram showing the relationship between a single example genome (i.e., the genotype) and the network that it instantiates (i.e., the phenotype). As indicated, each node gene indicates the functional role of a node and provides a unique identifier for that node (e.g., represented by integers in Fig. 2). Likewise, each connection gene indicates the identities of the two nodes that are being connected, the initial weight or strength (i.e., represented by the parameter ω) of the connection and whether or not that connection has been enabled (i.e., set equal to a

value of 0 or ω), and a unique identifier for that connection (again, as represented by the integers in Fig. 2). The unique mutation identifiers provide a numerical index that can be used to reference each new innovation that occurs across successive generations. (In the exposition below, this index will be called the *innovation number*.)

Figure 2

The complexity of the network topology increases over successive generations through genetic mutation. These mutations can affect both the number and type of nodes in a network, as well as the pattern of connectivity among the nodes. These two basic types of "connective" mutations are shown in Figure 3. New nodes are added to a network by replacing the connection between two existing nodes with an intermediary node that connects to the two original nodes (e.g., as shown in the left panel of Fig. 3). The new connections joining the new node to the original two are automatically enabled to ensure that the mutation affects the network's overall fitness. And similarly, mutations can affect the pattern of network connectivity by joining two previously unconnected nodes (e.g., as shown in the right panel of Fig. 3) or changing the strength of an existing connection between two nodes. The probability that each type of mutation (i.e., adding a node, adding a connection, or modifying a connection) will occur is determined probabilistically, with the overall rates of each respective type of mutation being controlled by the parameters $\pi_{add\text{-}node}$, $\pi_{add\text{-}link}$, and $\pi_{mutate\text{-}link}$ (for a complete list of the parameter values associated with macroscopic evolution, see Tables 1-5).

Figure 3

Tables 1-5

The precise manner in which connections are modified also depends upon the "severity" of a mutation, which is determined probabilistically using the parameter

$\delta_{severity}$. With probability $\delta_{severity}$, the mutation is considered to be "severe" and the connection is modified in one of the following mutually exclusive ways. With probability $c_{cold\text{-}gauss}$, the mutation in the existing connection is simply canceled, and with probability $1 - c_{gauss}$, the connection strength is sampled from a Gaussian distribution with $\mu_w = \omega$ (i.e., the initial weight value) and $\sigma_w = 0.5$. However, with probability $c_{gauss} - c_{coldgauss}$, the connection strength is sampled from a Gaussian distribution with $\mu_w = 0$ and $\sigma_w = 0.5$. The genes coding connections also have some probability, $c_{turn\text{-}on\text{-}off}$, of being turned on or off, so that connections that are enabled become disabled and vice versa. Finally, with the insertion of a new connection, there is some probability that two previously unconnected nodes cannot be found; when this happens, the parameter $\pi_{attempt\text{-}mutation}$ specifies the number of attempts that are made to locate the nodes before the effort is halted.

The problem of aligning two genomes during crossover is computationally inexpensive because innovation numbers provide a basis for rapidly comparing any two genomes. Those genes that are identical between two parent genomes are said to *match*, and those that do not match are in either *disjoint* or *excess*, depending on whether they occur within or outside (respectively) of the range of innovation numbers of the other parent's genome. Using such information, crossover between the genomes of any two parents can occur in three different ways, as shown in Figure 4.

Figure 4

In *single-point crossover*, an innovation number is randomly selected from the set of matching innovation numbers. All of the genes to the left of this point in one parent's genome (including disjoint genes but not excess genes because the latter by definition do not exist to the left of the crossover point) are then copied to the genome of the offspring, and all of the genes to the right of this point in the other parent's genome (including both disjoint and excess genes) are also copied to the genome of the offspring. If the gene at the crossover point happens to correspond to a connection weight, then that particular connection weight in the offspring is set equal to the mean of those connection weights in the two parents.

In *multipoint crossover*, each of the two possible values of all matching genes has an equal probability of being copied to the offspring's genome. And because all disjoint and excess genes can also be copied to the offspring's genome, the offspring's genome can become longer than the genome of either parent. This crossover method ensures that the offspring inherits both the shared and unique genes from both of its parents, thereby increasing both the combinatory and exploratory potential of this method relative to single-point crossover.

Finally, *multipoint-average crossover* is identical to multipoint crossover except that, rather than randomly assigning those matching genes representing the connection weights of one of the parents to the offspring, the offspring will instead inherit a gene for a connection weight that is the mean of the parents' two connection weights.

Each successive generation of networks is generated using some combination of the three aforementioned crossover methods in conjunction with both cloning and mutation. More specifically, with each new generation, individual offspring can be produced in three ways, determined in a probabilistic manner: (1) via cloning or copying the genome of a network from one generation to the next; (2) via mutating a genome (using one of the methods described above) and then copying the mutated genome to the next generation; and (3) via one of the three crossover methods that were described above. As will be indicated below, the genome of the fittest individual in each species is simply cloned from one generation to the next; each remaining individual has a probability equal to $p_{mutate\text{-}only}$ of having a mutation introduced into its genome (as described above) and that genome being copied to the next generation, and a probability of $1 - p_{mutate\text{-}only}$ of reproducing via crossover. With the latter, the method of crossover is determined probabilistically, with the probability of single-point, multipoint, and multipoint-average being specified by the parameters $p_{single\text{-}point}$, $p_{multipoint}$, and $p_{multipoint\text{-}average}$, respectively. Following crossover, the genome is left intact with probability $p_{mate\text{-}only}$ and a mutation is introduced to the genome with probability $1 - p_{mate\text{-}only}$. Finally, although crossover typically occurs between two individuals from the same species, it can with some small probability, $c_{inter\text{-}species}$, occur between two individual from different species, sometimes resulting in offspring

that are more fit than either parent.

Using these evolutionary methods, populations of networks having complex and diverse topologies can evolve. However, because networks having simpler topologies tend to optimize more rapidly than networks having more complex topologies, the process of adding nodes and connections initially causes more complex networks to be less fit. One method for preventing more complex network topologies from being prematurely removed from the population is to allow speciation, or the emergence of new "species" of networks (i.e., networks having more complex topologies) so that they can compete within their own more specialized ecological niches. By doing this, more complex networks are protected so that they have time to optimize their structures to their particular niches.

The process of speciation is also made computationally efficient by using the innovation numbers and by using the number of disjoint (D) and excess (E) genes as a metric of genome compatibility. The main intuition of this method is that pairs of genomes that contain large numbers of disjoint and/or excess genes are unlikely to share much of their evolutionary history, and are thus more likely to represent distinct species. This intuition is instantiated using a measure of the compatibility distance of two genomes. This distance, δ, is a linear combination of D, E, and $\overline{W}$, or the mean weight differences of matching genes, and is specified by Equation 1:

(1) $$\delta = \frac{c_1 D}{N} + \frac{c_2 E}{N} + c_3 \overline{W}$$

In Equation 1, c_1, c_2, and c_3 are coefficients to weight the relative contributions of the three relevant factors (i.e., excess genes, disjoint genes, and the mean weight difference of matching genes), and N is the number of genes in the longer of the two genomes and is used to normalize genome length. Any two genomes having a compatibility distance exceeding some pre-specified threshold, δ_c, are considered to belong to separate species and thus (usually) prohibited from breeding. In sorting genomes into species, each genome is grouped with the first species for which $\delta < \delta_c$, so that no genome belongs to more than one species.

Another potential problem associated with speciation is that one species can grow

without bounds, taking over the entire population. To prevent this from happening, the number of individuals within a species is increased/decreased according to whether its total fitness is above/below the mean fitness of the population of species. This is done using Equation 2:

$$(2)\quad N_j^{'} = \frac{\sum_{i=1}^{N_j} f_{i,j}}{\bar{f}}$$

where $N_j^{'}$ is the number of individuals in species j adjusted for global mean fitness, as specified by the right side of the equation. There, N_j is the non-adjusted number of individuals in species j, $f_{i,j}$ is the adjusted (relative) fitness of individual i of species j (as given by Equation 3), and $\bar{f}$ is the mean fitness (i.e., the mean of $f_{i,j}$) of the entire population.

$$(3)\quad f_{i,j} = \frac{f_{i,j}^{raw} - f_{worst}}{\sqrt{N_j}}$$

In Equation 3, $f_{i,j}^{raw}$ is the raw fitness of individual i of species j (which normal takes on a negative value) and f_{worst} is the worst raw fitness in the population. Equation 3 differs from the original *NEAT* algorithm (Stanley & Miikkulainen, 2002) in that the denominator is the square root of N_j rather than N_j to take advantage of fitter species. The size of each species is thus adjusted so that the number of individuals in more fit species tends to increase and the number of individuals in less fit species tends to decline. With each new generation, a certain percentage (determined by the parameter $c_{survival}$) of the best performing (i.e., most fit) of each species is allowed to randomly produce the next generation of for their species.

A fitness amplification assumption is also added to the original *NEAT* algorithm that allows the fittest individual in the population to have the unique opportunity to contribute a copy of its genome (i.e., a clone of itself) to the next generation. This is done using Equation 4, where f_{best} is the adjusted fitness value of the best-performing member of a species, and c_{best} is an amplification coefficient that enhances that fitness value so as to ensure that that individual contributes a copy of itself to the next generation.

(4) $f_{best} \leftarrow c_{best} f_{best}$

To further enhance the reproductive advantage of better species and thereby increase the probability of evolution being successful, a "delta coding" procedure is introduced that "steals" or eliminates a certain number of offspring, $d_{offspring\text{-}stolen}$, from whatever species has improved the least. In a similar manner, when the least fit species has not improved over a certain number of generations, $d_{drop\text{-}off\text{-}age}$, its overall fitness is reduced; to protect younger species, their fitness is increased by an amplification factor, $d_{age\text{-}signifance}$.

It is worth emphasizing that the algorithm as described so far is biased to favor networks having simple topologies. The reason for this is that the addition of nodes and/or connections is not without cost: As the topology of a network increases, so too does the inherent difficult associated with evaluating its contribution to a network's fitness. Thus, by starting the evolutionary process with the simplest possible networks (i.e., networks without hidden units), more complex topologies are only retained if they are justified on the grounds that they increase a network's fitness.

Finally, the probability of mutation is determined by a *simulated annealing* process (Černý, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983) in which the overall probability is initial some large value but then declines with each successive generation. (The "simulated annealing" metaphor comes from metallurgy, were the initial high temperature of a metal is slow decreased over time so that the atoms can align in a way that ensures high tensile strength.) The rate of this decline is controlled by a "temperature" parameter T, and the probabilities of adding a new node, $\pi_{add\text{-}node}$, or a new connection link, $\pi_{add\text{-}link}$, to a network are given by Equations 5 and 6, respectively:

(5) $\pi_{add-node} = \max\left[\psi_1, \min\left(\psi_2, x\right)\right]$, where $x \leftarrow x - \frac{1}{k_1 T}$

(6) $\pi_{add-link} = \max\left[\psi_3, \min\left(\psi_4, x\right)\right]$, where $x \leftarrow x - \frac{1}{k_2 T}$

where Ψ_1 and Ψ_2 respectively represent the lower and upper limits for the probability of adding a node via mutation, Ψ_3 and Ψ_4 respectively represent the lower and upper

limits for the probability of adding a link via mutation, and k_1 and k_2 represent coefficients for adjusting the probabilities of adding nodes and links, respectively. Similarly, in Equation 7, Ψ_5 and Ψ_6 respectively represent the lower and upper limits for the probability of reproduction via mutation only, Δ is a parameter that control how much the probability is incremented per generation, and k_3 is a dynamic coefficient that is set equal to -1 if the fitness of the population fails to improve by some criterion, $c_{annealing}$ (otherwise, k_3 is set equal to 1). Finally, if the fitness of the population fails to improve, the values of the $\pi_{add\text{-}node}$ and $\pi_{add\text{-}link}$ are set equal to the starting values.

(7) $p_{mutate-only} = \max\left[\psi_5, \min\left(\psi_6, x\right)\right]$, where $x \leftarrow x + k_3\Delta$

Microscopic evolution. Our algorithm for microscopic evolution of network connection weights is based on the *Covariance Matrix Adaptation Evolution Strategy* (*CMA-ES*; Hansen, 2006; Hansen & Kern, 2004; Hansen et al., 2003; Hansen & Ostermeier, 2001; Igel et al., 2007; Suttorp et al., 2009) and provides a stochastic method for parameter optimization of non-linear, non-convex functions. As such, it is particularly useful for "rugged" parameter landscapes comprised of discontinuities and local optima (e.g., sharp "ridges") and is thus well suited to solve ill-conditioned and non-separable problems. Some modifications of the CMA-ES algorithm were necessary, however, to make it more amenable to the problem of evolving network connection weights.

The general intuition behind the algorithm is that, rather than training a network across a series of trials to find the set of connection weights that allow a network to solve some problems, the connection weights are instead evolved. To understand how this is done, it is first necessary to understand that the weights themselves can be represented as a vector, and that the elements of this vector can be sampled to find values that allow a network to solve a particular problem. In essence, this is what the CMA-ES algorithm does: During each learning trial, a set of vectors representing possible connection weight solutions to the problem are sampled from a sampling distribution, and then a new mean and covariance of a sampling distribution are

computed that reflect the fitness of these sampled vectors. This whole process is repeated until either a solution meeting some goodness-of-solution criterion has been reached, or a stopping criterion has been reached.

During each trial $t + 1$, the λ individual vectors (i.e., candidate solutions consisting the connection weights) are sampled using Equation 8:

(8) $$x_k^{t+1} \sim N\left(\langle x\rangle_w^{(t)}, \sigma^{2(t)} C^{(t)}\right), k = 1,...,\lambda$$

where $N\left(\langle x\rangle_w^{(t)}, \sigma^{2(t)} C^{(t)}\right)$ is a normally distributed vector with mean $\langle x\rangle_w^{(t)}$, sampling variance $\sigma^{2(t)}$, and covariance matrix $C^{(t)}$. (For a complete list of the parameter values associated with microscopic evolution, see Tables 6-10.) This provides a simple method for sampling candidate vectors of connection weights using a multi-normal distribution, with the covariance matrix determining the degree to which sampling proceeds in a cautious versus audacious manner.

Tables 6-10

The mean of the sampling distribution is computed using the weighted average of the individual sampled vectors using Equation 9:

(9) $$\langle x\rangle_w^{(t)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(t)}$$

with the constraints that $w_i > 0$ for all values of i and $\sum_{i=1}^{\mu} w_i = 1$, and with the index $i{:}\lambda$ denoting the i-th best individual. By combining the best connection weight vectors in this weighted manner, the mean of the sampling distribution approaches the target solution over time.

Returning to Equation 8, $C^{(t)}$ represents the correlation between different connection weights within a neural network, and $\sigma^{2(t)}$ modulates the variability associated with the sampling distribution. Both terms are therefore vital to the success of the algorithm. Over trials, $C^{(t)}$ changes as a function of both the *evolutionary path*, which as its name suggests, controls the global orientation or trajectory of the evolutionary process, thereby allowing it to converge towards a solution. $C^{(t)}$ is updated using Equation 10, were c_{cov} is a parameter that controls the rate of change,

μ_{cov} is a parameter for weighting between the rank-one and rank-μ update, and the terms $P_c^{(t+1)}$ and $S^{(t+1)}$ are specified by Equations 11 and 12, respectively.

$$(10)\quad C^{(t+1)} = \left(1 - c_{\text{cov}}\right)C^{(t)} + c_{\text{cov}}\frac{1}{\mu_{\text{cov}}}P_c^{(t+1)} + c_{\text{cov}}\left(1 - \frac{1}{\mu_{\text{cov}}}\right)S^{(t+1)}$$

$$(11)\quad P_c^{(t+1)} = p_c^{(t+1)}\left[p_c^{(t+1)}\right]^T$$

$$(12)\quad S^{(t+1)} = \sum_{i=1}^{\mu}\frac{w_i}{\sigma^{2(t)}}\left[x_{i:\lambda}^{(t+1)} - \langle x\rangle_w^{(t)}\right]\left[x_{i:\lambda}^{(t+1)} - \langle x\rangle_w^{(t)}\right]^T$$

In Equation 11, the term $p_c^{(t+1)}$ is specified by Equation 13, which accumulates the differences between the mean connection-weight sampling distribution vectors across successive trials:

$$(13)\quad p_c^{(t+1)} = \left(1 - c_c\right)p_c^{(t)} + H_\sigma^{(t+1)}\sqrt{c_c\left(2 - c_c\right)}\frac{\sqrt{\mu_{eff}}}{\sigma^{(t)}}D^{(t+1)}$$

where c_c is learning rate for accumulation for the rank-one update of the covariance matrix, and $H_\sigma^{(t+1)} = 1$ if $\frac{\left\|p_\sigma^{(t+1)}\right\|}{\sqrt{1-\left(1-c_\sigma\right)^{2\tau}}} < \left(1.4 + \frac{2}{n+1}\right)E\left(\left\|N(0,I)\right\|\right)$; otherwise, $H_\sigma^{(t+1)} = 0$. (In the preceding conditional statement, the index τ denotes the number of completed trails.) The term μ_{eff} is the *variance effective selection mass*, which is related to recombination weights and is constrained so that $\mu_{eff} = 1\Big/\sum_{i=1}^{\mu} w_i^2$. Finally, the last term in Equation 13 is the actual difference between mean sampling distribution vectors across successive trials and is specified by Equation 14:

$$(14)\quad D^{(t+1)} = \left(\langle x\rangle_w^{(t+1)} - \langle x\rangle_w^{(t)}\right)$$

Finally, the sampling variance in Equation 8, $\sigma^{2(t)}$, controls the overall rate of change in the evolutionary process. The value of $\sigma^{2(t)}$ during any given trial is given by Equation 15, in which the term is specified by Equation 16.

$$(15)\quad \sigma^{(t+1)} = \sigma^{(t)}\exp\left[\frac{c_\sigma}{d_\sigma}\left(\frac{\left\|p_\sigma^{(t+1)}\right\|}{E\left(\left\|N(0,I)\right\|\right)} - 1\right)\right]$$

$$(16)\quad p_\sigma^{(t+1)} = \left(1 - c_\sigma\right)p_\sigma^{(t)} + \sqrt{c_\sigma\left(2 - c_\sigma\right)}B^{(t)}E^{-1(t)}B^{T(t)}\frac{\sqrt{\mu_{eff}}}{\sigma^{(t)}}D^{(t+1)}$$

In Equation 16, the orthogonal matrix, $B^{(t)}$, and the diagonal matrix, $E^{(t)}$, are obtained via principal component analysis of $C^{(t)}$ using the matrix theorem: $C^{(t)} = B^{(t)} E^{2(t)} B^{T(t)}$. In Equation 15, $E\left(\left\|N(0,I)\right\|\right)$ is the expected length of p_σ under random selection and is given by Equation 17:

$$(17)\quad E\left(\left\|N(0,I)\right\|\right) = \frac{\sqrt{2}\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \approx \sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right)$$

where n denotes the number of search-space dimensions, which in this application simply corresponds to the number of connection weights.

As indicated above, our algorithm modifies the standard CMA-ES algorithm (as described so far) to automatically inflate the population size, λ, and the number of trial iterations for the stopping condition, τ_{stop}, whenever the algorithm *stagnates*, or stops converging towards a better solution. This is done using Equations 18 and 19, respectively, where o denotes the number of generations over which the algorithm has stagnated, and ρ is a parameter that denotes the default number of generations.

$$(18)\quad \lambda = 4 + \left[3\ln(n+o)\right]$$

$$(19)\quad \tau_{stop} = \rho(1+o)$$

Furthermore, to improve microscopic evolution, especially when the population is trapped in local maxima, the sampling variance in initial trial, $\sigma^{(0)}$, is defined according to its stagnation level using Equation 20. By increasing the sample, the population may thereby escape the local maxima point.

$$(20)\quad \sigma^{(0)} = \min\left[\sigma_{\max}, \sigma_d + o_\sigma \ln(o+1)\right]$$

where σ_{max} represents the upper limit of initial sampling variance, σ_d represents the default sampling variance, and o_σ represents a coefficient for scaling the effect of stagnation.

Finally, a fitness criterion can also be used to stop the evolutionary process. That is, if the fitness of the network, f, exceeds some fitness threshold, f_{stop}, or the trial iteration exceeds some criterion, τ_{stop}, then the microscopic evolution of the

connection weights is halted.

Reinforcement Learning. Reinforcement learning refers to a general class of machine-learning algorithms in which performance is "shaped" using a single training signal corresponding to the reward/punishment that is associated with specific actions and/or the states that result from those actions (Sutton & Barto, 1998). Of central importance to this notion is the idea of *reward prediction error*, usually denoted by δ, which represents the reward that an artificial agent anticipates in response to a particular action and the state that then results from that action. This is represented in Equation 21, in which R represents the immediate reward that is received from executing the action, x represents a particular state that the agent can be in at time t, and the *value function* V represents the value associated with the state. The parameter γ is a *discount* parameter that determine how much the reward that is anticipated to result from the next state, x_{t+1}, is weighed against the immediate reward; small values of γ thus make the agent "greedy" in that it tends to prefer actions that result in large immediate rewards, whereas large values of γ cause the agents to prefer actions that result in rewards over the long run. (For a complete list of the parameter values associated with reinforcement learning, see Table 11.)

$$\delta = R + \gamma V\left(x_{t+1}\right) - V\left(x_t\right) \tag{21}$$

One way to implement this algorithm within a neural network is to use the standard error-driven back-propagation algorithm (Rumelhart, et al., 1986), training the network using each state at time t, x_t, as the input, and allowing the resulting state at time t+1, x_{t+1}, as the output. The teaching signal for the desire output is then $R + \gamma V(x_{t+1})$, and the weight for any given connection in the network is adjusted during learning using Equation 22:

$$\Delta w_{direct} = \alpha\left[R + \gamma V\left(x_{t+1}\right) - V\left(x_t\right)\right]\frac{\partial V\left(x_t\right)}{\partial w} \tag{22}$$

where α is a learning rate parameter that controls the rate of convergence and the subscript "direct" denotes the name of this algorithm. Although this *direct algorithm* has been used successfully in many applications (Tesauro, 1990, 1992), it is not

guaranteed to converge for general function-approximation systems (Baird, 1995).

To develop such an algorithm, the problem can be restated as being one of predicting the outcome of a deterministic Markov chain, with the goal being to specify a value function that, for any given state, x_t, will give the value of the immediate reward and the successor state, x_{t+1}, thereby satisfying the *Bellman equation* (Bellman, 1957):

(23) $$V(x_t) = \langle R + \gamma V(x_t) \rangle$$

where <> is the expected value of all possible successor states, x_{t+1}. For a system having a finite number of states, the optimal value function, V^*, will provide a unique solution to the above equation, and any value function that is suboptimal will result in an inequality called the *Bellman residual*. For a system with n states, the mean squared Bellman residual is given by Equation 24, and provides a direct measure, E, of the degree to which a given policy is suboptimal. And because the value of E is bounded, it suggests an alternative to the direct algorithm for adjusting the connection weights in a neural network function approximation system: by performing stochastic gradient descent on E.

(24) $$E = \frac{1}{n} \sum_x \langle R + \gamma V(x_{t+1}) - V(x_t) \rangle^2$$

Under the assumption that V is parameterized by the set of connection weights, the adjustment to any given weight w following a transition from x_t to x_{t+1} with reward R is specified by Equation 25:

(25) $$\Delta w_{residual-gradient} = -\alpha \left[R + \gamma V(x_{t+1}) - V(x_t) \right] \left[\gamma \frac{\partial}{\partial w} V(x_{t+1}) - \frac{\partial}{\partial w} V(x_t) \right]$$

where the "residual-gradient" subscript denotes the name of the algorithm, residual-gradient. For a system with a finite number of states, E will equal 0 only if V = V*. And critically, this residual-gradient algorithm is guaranteed to converge, thus making it ideally suited for training neural networks to be function-approximation systems. The one limitation of this algorithm, however, is that it is slow (Baird, 1995; Williams & Baird, 1993). This limitation makes the algorithm impractical for large-scale problems of the type that might be of interest to psychologists (e.g.,

Reichle & Laurent, 2006). This limitation results in the following quandary: Whereas the direct algorithm is rapid, it is not guaranteed to converge, especially for large problems; in contrast, the residual-gradient algorithm is guaranteed to converge, but is slow, especially for large problems. Because both algorithms are based on gradient descent, however, the solution to this problem is fairly straightforward: One simply combines the two algorithms so that the adjustment to any given connection weight *w* is simply some weighted average of the adjustments given by the direct and residual-gradient algorithms. This is done using Equation 26:

$$\Delta w_{residual} = (1-\phi)\Delta w_{direct} + \phi \Delta w_{residual-gradient} \quad (26)$$

where the subscript "residual" denotes that this is the *residual algorithm*. In the equation, the parameter ϕ modulates the degree to which the direct and residual gradient algorithms contribute to the adjustment of a given weight, *w*. Our algorithm thus benefits from the strengths of the direct and residual-gradient algorithms by combining the speed of the former with the convergence of the latter.

References

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning (ICML95)* (pp. 30-37). San Mateo, CA: Morgan Kaufman.

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, 45*, 41-51.

Hansen, N. (2006). The CMA evolution strategy: A comparing review. In J. A. Lozano, P. Larrañaga, I. Inza, & E. Bengoetxea (Eds.), *Towards a New Evolutionary Computation . Advances in estimation of distribution algorithms* (Vol. 192, pp. 75-102). Berlin: Springer.

Hensen, N. & Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullianaria, J. Rowe, P. Tino, A. Kabán, & H.-P. Schwefel (Eds.), *Eighth International Conference on Parallel Problem Solving from Nature* (*PPSV VIII*), *Vol. 3242* (pp. 282-291). Berlin: Springer.

Hansen, N., Müller, S. D., & Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation, 11*, 1-18.

Hansen, N. & Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation, 9*, 159-195.

Igel, C., Hansen, N., & Roth, S. (2007). Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation, 15*, 1-28.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671-680.

Reichle, E. D. & Laurent, P. A. (2006). Using reinforcement learning to understand the emergence of "intelligent" eye-movement behavior during reading.

Psychological Review, 113, 390-408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by backpropagating errors. *Nature, 323*, 533-536.

Stanley, K. O. & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*, 99-127.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Suttorp, T., Hansen, N., & Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning, 75*, 167-197.

Tesauro, G. (1990). Neurogammon: A neural-network backgammon program. *Proceedings of the International Joint Conference on Neural Networks* (vol.3, pp. 33-40). San Diego, CA: IEEE Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*, 257-277.

Williams, R. J., & Baird, L. C. (1993). *Tight performance bounds on greedy policies based on imperfect value functions*. Northeastern University Technical Report NU-CCS-93-14, November.

Table 1: Parameters controlling mutation.

Parameter	Description	Value
$c_{cold\text{-}gauss}$	probability of canceling weight mutation in existing connection	0.1, if $\delta_{severity}$ = true; 0.3, otherwise
c_{gauss}	= 1 – probability of sampling new connection from distribution having the same mean connection strength	0.3, if $\delta_{severity}$ = true; 0.5, otherwise
$c_{turn\text{-}on\text{-}off}$	probability of turning on/off the selected connection gene	0.2
$\delta_{severity}$	probability of "severe" connection weight mutation	0.5
$\pi_{add\text{-}link}$	Probability of adding connection	0.6
$\pi_{add\text{-}node}$	Probability of adding node	0.2
$\pi_{attempt\text{-}mutation}$	Number of attempts to locate "lost" nodes	50
$\pi_{mutate\text{-}link}$	Probability of connection mutation	0.9
σ_w	Standard deviation of mutated connection distribution	0.5

Table 2: Parameters controlling the reproduction of offspring.

Parameter	Description	Value
c_{best}	fitness amplification coefficient for best genome	3
$c_{inter\text{-}species}$	probability of inter-species mating	0.2
$c_{survival}$	percent of genome per species that survives	0.2
p	population size	120
$p_{mate\text{-}only}$	probability of reproducing only via mating	0.2
$P_{multipoint}$	probability of multipoint crossover	0.6
$p_{multipoint\text{-}average}$	probability of multipoint-average crossover	0.4
$p_{mutate\text{-}only}$	probability of mutating only	0.3
$p_{single\text{-}point}$	probability of single-point crossover	0.3

Table 3: Parameters controlling the compatibility distance metric.

Parameter	Description	Value
c_1	scaling factor for disjoint genes	1
c_2	scaling factor for excess genes	1
c_3	scaling factor for gene differences	2
δ_c	minimal distance for 2 genomes to be in same species	3

Table 4: Parameters controlling the "delta coding" procedure.

Parameter	Description	Value
$d_{age\text{-}significance}$	age amplification factor	1
$d_{drop\text{-}off\text{-}age}$	generations after which poor species are penalized	2000
$d_{offspring\text{-}stolen}$	number of offspring eliminated from unfit species	10

Table 5: Parameters controlling simulated annealing.

Parameter	Description	Value
$c_{annealing}$	criterion for simulated annealing	10
Δ	increment for adjusting $p_{mutate\text{-}only}$	0.01
k_1	coefficient for adjusting probability of adding node	20
k_2	coefficient for adjusting probability of adding link	10
ψ_1	lower probability limit of adding node	0.02
ψ_2	upper probability limit of adding node	0.04
ψ_3	lower probability limit of adding link	0.1
ψ_4	upper probability limit of adding link	0.2
ψ_5	lower limit of $p_{mutate\text{-}only}$	0.3
ψ_6	upper limit of $p_{mutate\text{-}only}$	0.5

Table 6: Parameters controlling the initial settings.

Parameter	Description	Value
$C^{(0)}$	covariance matrix	I (i.e., identity matrix)
$p_c^{(0)}$	evolutionary path for covariance matrix	0
$p_\sigma^{(0)}$	evolutionary path for variance	0
σ_{max}	upper limit for sample variance	1.0
σ_d	default value for sample variance	0.5
o_σ	coefficient for scaling stagnation effect	0.01

Table 7: Parameters controlling the Covariance Matrix Adaptation.

Parameter	Description	Value
c_c	learning rate for accumulation for rank-1 update of the covariance matrix	$4 / (n+4)$
c_{cov}	initial learning rate for covariance matrix update	$\frac{1}{\mu_{\text{cov}}}\frac{2}{\left(n+\sqrt{2}\right)^2}+\left(1-\frac{1}{\mu_{\text{cov}}}\right)\min\left(1,\frac{2\mu_{eff}-1}{(n+2)^2+\mu_{eff}}\right)$
μ_{cov}	Weighting between rank-1 and rank-m update	μ_{eff}

Table 8: Parameters controlling the step size.

Parameter	Description	Value
c_σ	learning rate for accumulation of step-size control	$\frac{\mu_{eff}+2}{n+\mu_{eff}+3}$
d_σ	damping for step-size update	$1+2\max\left(0,\sqrt{\frac{\mu_{eff}-1}{n+1}}-1\right)+c_\sigma$

Table 9: Parameters controlling selection and recombination.

Parameter	Description	Value
λ	candidate sample size	4+[3 ln(n+o)]
μ	parent size	$\lambda/2$
μ_{eff}	variance effective selection mass	$\frac{1}{\sum_{i=1}^{\mu} w_i^2}$
$w_{i=1\ldots\mu}$	Recombination weights	$\frac{\ln(\mu+1)-\ln(i)}{\sum_{j=1}^{\mu}\left[\ln(\mu+1)-\ln(j)\right]}$

Table 10: Parameters controlling the stopping conditions.

Parameter	Description	Value
f_{stop}	fitness stopping criterion	best fitness in history
ρ	default stopping coefficient	1000

Table 11: Initial parameters for the reinforcement learning of network connection weights.

Parameter	Description	Value
α	learning rate	0.05
γ	discount rate	0.9
ϕ	balance rate	0.5

Figure Captions

Figure 1. A schematic diagram of the evolutionary process, in its entirety. The top box shows the population, which consists of *N* species of genotypes and their resulting phenotype networks. Each generation is trained on a problem using the residual-gradient reinforcement-learning algorithm. The micro-evolution (e.g., *CMA-ES* algorithm) is then used to sample network connection weights, so as to "nudge" the improvement of overall fitness. The reproduction procedure is allowed to produce the next generation via mutation, crossover, or cloning, as determined probabilistically.

Figure 2. An example showing the coding and mapping of a single genotype (in the top panel) onto its corresponding phenotype, an artificial neural network (in the bottom panel). As indicated, the genotype codes node identity and type, as well as information about connections between nodes, including their pattern of connectivity, weight, innovation number, and whether or not they are enabled.

Figure 3. An example illustrating two types of mutations: On the left, the mutation results in the addition of a node, while on the right, the mutation results in the addition of a connection weight between two nodes.

Figure 4. The top panel shows the genomes of two parents, along with the corresponding networks. The middle panel shows the two genomes aligned, as would occur during crossover. The bottom panel shows the three different crossover methods (single-point, multipoint, and multipoint-average) and the resulting offspring phenotypes.

Figure 1.

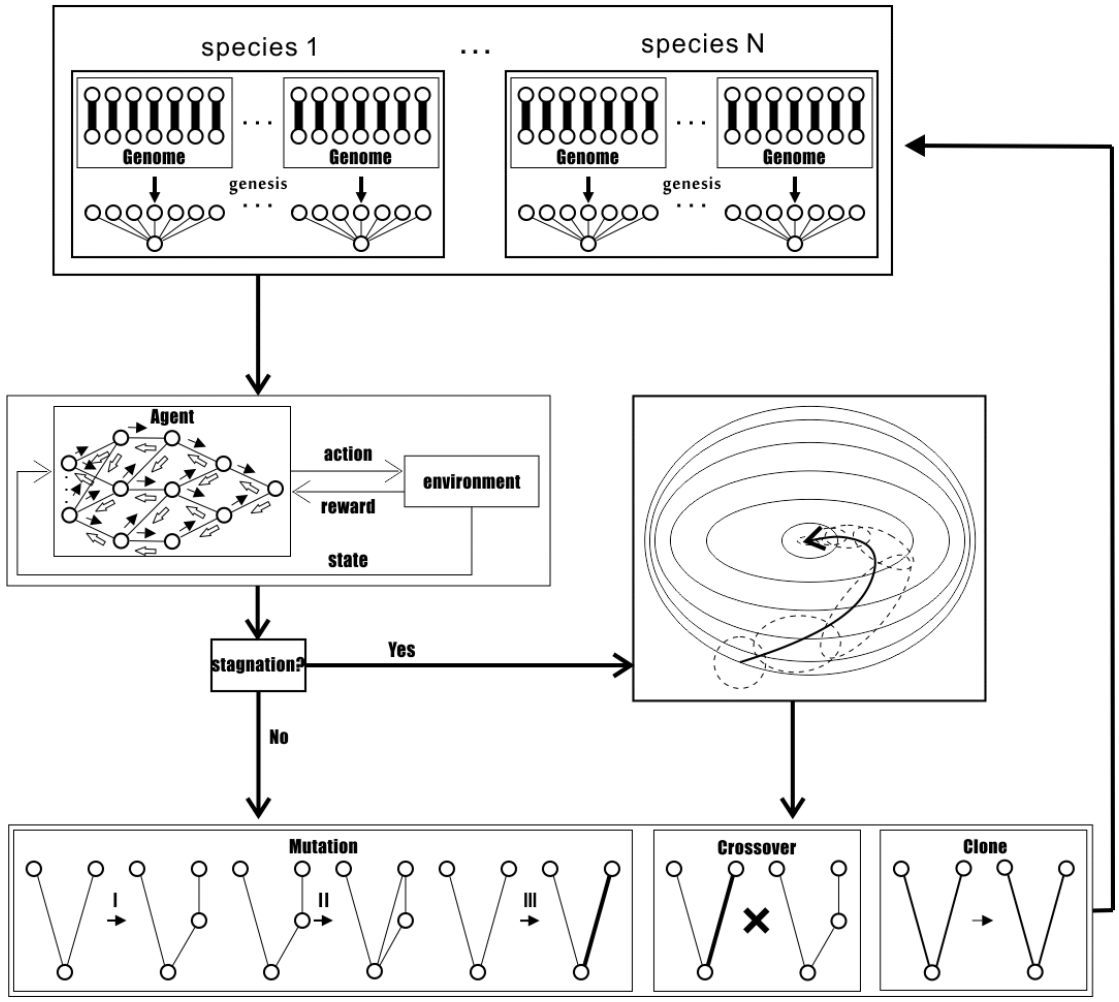

species 1
...
species N
Genome
Genome
Genome
Genome
genesis
genesis
Agent
action
environment
reward
state
stagnation?
Yes
No
Mutation
I
II
III
Crossover
Clone

Figure 2.

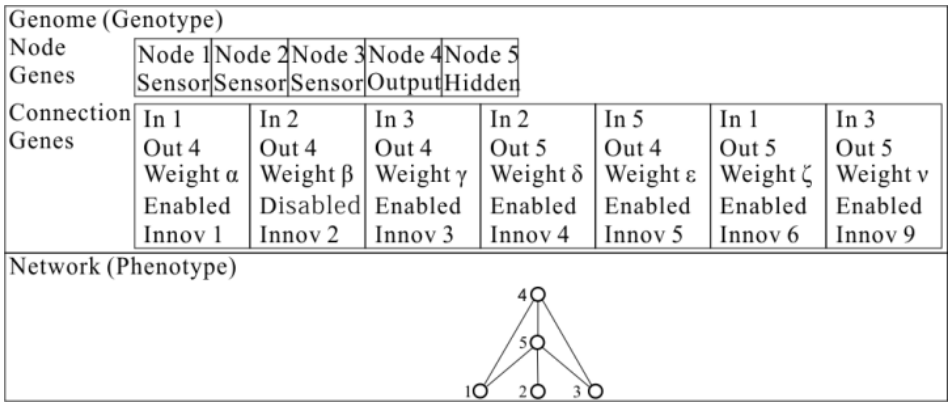

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connection Genes
In 1 Out 4 Weight α Enabled Innov 1
In 2 Out 4 Weight β Disabled Innov 2
In 3 Out 4 Weight γ Enabled Innov 3
In 2 Out 5 Weight δ Enabled Innov 4
In 5 Out 4 Weight ε Enabled Innov 5
In 1 Out 5 Weight ζ Enabled Innov 6
In 3 Out 5 Weight ν Enabled Innov 9
Network (Phenotype)
4
5
1
2
3

Figure 3.

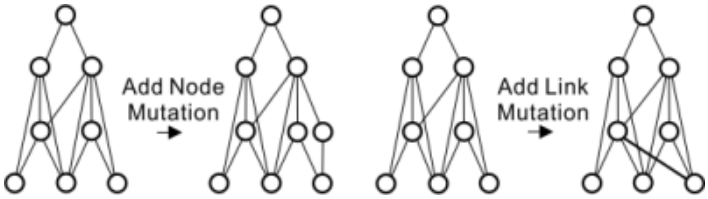
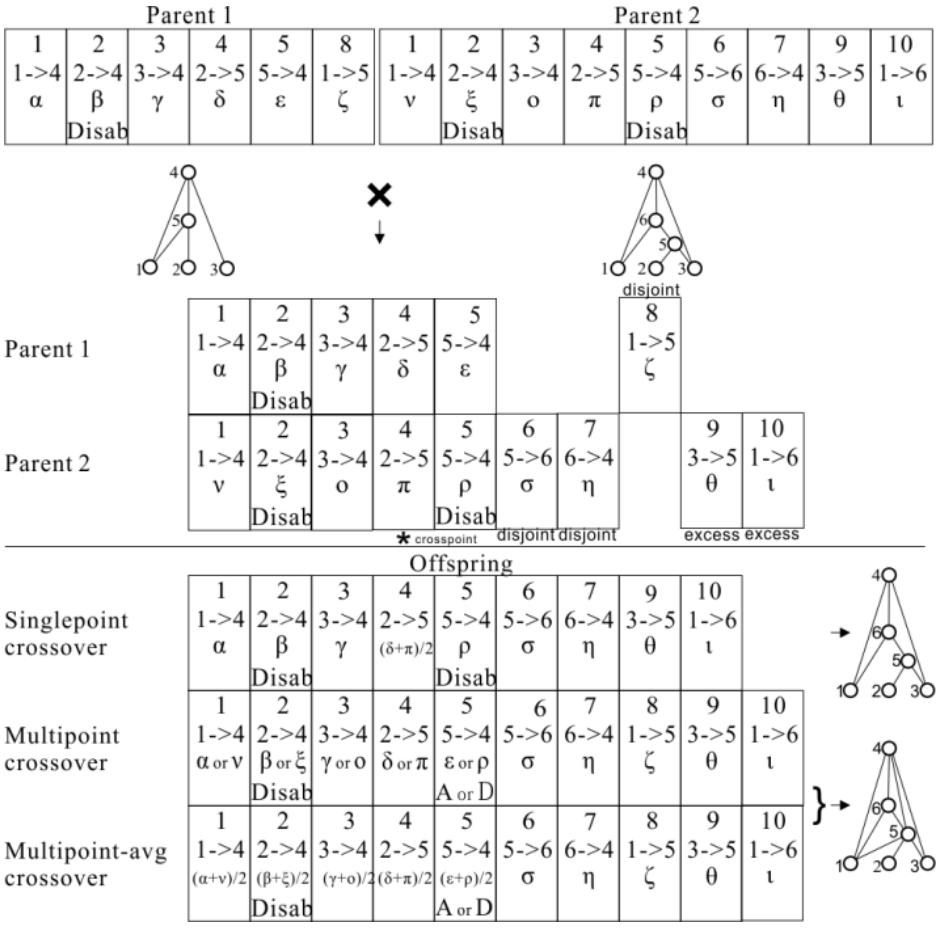


Figure 4.


Parent 1
Parent 2
1 1->4 α
2 2->4 β Disab
3 3->4 γ
4 2->5 δ
5 5->4 ε
8 1->5 ζ
1 1->4 ν
2 2->4 ξ Disab
3 3->4 ο
4 2->5 π
5 5->4 ρ Disab
6 5->6 σ
7 6->4 η
9 3->5 θ
10 1->6 ι
disjoint
crosspoint
disjoint disjoint
excess excess
Offspring
Singlepoint crossover
Multipoint crossover
Multipoint-avg crossover
(δ+π)/2
α or ν
β or ξ
γ or ο
δ or π
ε or ρ
A or D
(α+ν)/2
(β+ξ)/2
(γ+ο)/2
(δ+π)/2
(ε+ρ)/2