Combining Evolutionary Algorithms and Neural Networks

Keith L. Downing
The Norwegian University of Science and Technology
Trondheim, Norway
keithd@idi.ntnu.no

September 27, 2006

1 Introduction

Returning to the robot example of the earlier chapters, note that the various EA solutions had one key feature in common: they evolved strategies that did not change during the simulated lifetime of the phenotypes. In short, the phenotypes did not learn.

In real-world situations, hard-wired solutions, whether evolved using EAs or hand-coded by humans, are difficult to trust due to the unpredictable nature of real-world environments. A strategy designed for sunny days may become disastrous in the rain, or one that assumes a flat surface is thrown into total confusion by a slight incline.

In general, a good many situations call for AI systems that can adapt to their surroundings. Evolution is one form of adaptation, but it typically runs at a slower time scale than that of environmental change. Hence, the individuals produced by evolution must also have plasticity: they must be capable of changing their behavior to tackle unexpected conditions.

Thus, it makes good sense to combine evolutionary algorithms and artificial neural networks into evolving artificial neural networks (EANNs). One EANN, Cellular Encoding, was described in the EA applications chapter; there are many more. Particularly in the field of Evolutionary Robotics, researchers have long recognized the power of EANNs.

Originally, EANNs were designed to **replace** learning with evolution, not to integrate the two mechanisms. Basically, many situations call for ANNs with memory, which requires recurrent links. Unfortunately, recurrent topologies make classic supervised learning via backpropagation very difficult. Disregarding recurrence, supervised learning is still inappropriate for most robotic applications, since omniscent teachers cannot provide the *correct* answer/response for each situation, particularly when a typical robot samples the environment (via it sensors) and reacts several times per second.

Robotic applications are similar to many game playing situations in which feedback comes only at the end of the task; a teacher cannot evaluate and correct each individual move. Such tasks are amenable to reinforcement learning, while basic pattern clustering aspects of the problem can call for unsupervised learning. But supervised learning of correct moves is normally infeasible.

In general, the ANN architecture is very useful for mapping from sensory inputs to motor outputs, and the addition of recurrence allows memories to also affect action choices; but learning the correct weights on these links is non-trivial. One popular solution is an EANN in which the weights are determined by evolution. These systems do not learn, per say, since the weights are fixed during the phenotype's lifetime. But at least the EA approach can find some, very useful, weight vectors for ANNs, and it works equally well for both strictly-feed-forward and recurrent networks.

2 Three Levels of Adaptivity

In reviewing a small sampling of the EANN work, we give special focus to the level(s) of adaptivity included in a system. These are:

1. Phylogenetic or Evolutionary - Characterized by the use of an EA and thus having clearly definable genotypic and phenotypic levels, genetic operators, fitness functions, etc.
2. Ontogenetic or Developmental - Involving a non-trivial genotype-to-phenotype translation. In most cases, the genotype is a *recipe* that, through some recursive growth process, produces the phenotype.
3. Epigenetic or Learning - During actual performance testing, the system is able to modify itself in some manner that effects future behavior.

AI systems that include all three are known a POE (Phylogenetic, Ontogenetic and Epigenetic) or TRIDAP (three-way adaptive) systems. However, only a small sampling of AI systems include all 3 of these mechanisms. Most researchers find that one or two adaptive levels are sufficient for their problems.

In natural systems, the border between evolution and both development and learning is relatively easy to delineate, but the latter two overlap considerably.

Although evolution is an expansive, continuous and continual process, it can be described as a repeated sequence of discrete, overlapping activities: living and reproducing. Reproduction is singled out (among all the other life processes) since only it directly affects evolution's course. Each new round of genotypes represents another attempt at solving the survival problem.

Embryogeny begins directly after reproduction, with no significant overlap between the two. Hence, development immediately follows the latest shift in evolutionary direction (i.e., the latest change to the genotype pool). It then embodies an essential piece of the life process, and one that begins the search in this new direction that evolution has chosen.

Epigenesis, (a.k.a., plasticity or learning), involves environmentally-initiated adaptive changes to the post-natal organism. However, the line between plasticity and development frequently blurs, often in concert with the popular nature-versus-nurture debates. Although many life processes, such as growth, seem predominately governed by genes and post-natal development, they are easily derailed by an overly-restrictive (or overly-permissible) environment. Similarly, plastic changes such as increased muscle mass due to athletic training, are constrained by an individual's genetic code.

Neuroscience only adds further confusion to the development-learning distinction. One useful distinction posits neuron formation, migration and death, along with axonal and dendritic growth, as developmental processes, while the fine-tuning of synapses is the neurological essence of learning. However, all of these processes occur to greater and lesser degrees during embryogeny, post-natal development, and even adulthood.

Since nature so seamlessly interleaves development and plasticity, any clean temporal separation between them appears largely artificial.

Fortunately, as the designers of biologically-inspired computer systems, we stand free to embrace or ignore nature's spatial and temporal contexts of development. Our boundaries may be fuzzy, while her's are strict, or vice versa.

In the case of EANNs, we define development as all aspects of phenotypic formation (and modification) that occur **prior to** the beginning of fitness testing. Then, any changes to the phenotype that occur **during** fitness testing constitute learning.

With EANNs, development (if any occurs) typically produces the network topology (i.e., the neurons and their connection pattern), while learning involves the changing of connection weights due to feedback during performance testing.

3 Example EANNs

Montana and Davis [9] were the first to use EAs to learn weight vectors for ANNs. Their task was actually a classic supervised-learning task: classification of patterns, in their case, patterns in sonar data. Their genotype was a simple list of real numbers, each representing one weight of the ANN. These were mutated by the addition of small increments (in the range -1 to 1). Crossover points were restricted to chromosomal locations that were boundaries between the input weight sets of individual neurons (i.e., the weights on all incoming links to a neuron). These sets of input weights are common building blocks in EANNs, so their crossover operator was biased to treat these as atomic units.

Their system out-performed standard backpropagation on the supervised learning task, but it was beaten by newer, optimized versions. They therefore concluded that the approach was useful for supervised learning but probably best for unsupervised situations.

Normally, to test the performance of a neural network, the data set is split into two parts: a training set, s_1 and a test set, s_2. Members of s_1 are then sent through the network many (hundreds or thousands of) times in order to gradually adjust the weights to minimize output error. When training is complete, the members of s_2 are run through the network **just once**, and the total error is recorded. Low error indicates that the network has generalized from its experience with s_1, i.e., that it has learned a general concept and not simply memorized the training cases.

With EANNs that evolve weights, there is no need for separate data sets. The weights are essentially guessed during the initialization of the first generation, filtered by selection and refined by genetic operators over many generations. They are not learned in any one generation. Fitness is based on the accuracy of the network in classifying each member of the data set, so after many generations, this accuracy increases.

Another one of the early EANN systems, by Miller et. al. [8], used evolution to determine the connection topology of the network. Essentially, the genotype was a binary array in which the rows were laid side by side to form one long bit vector. The array, a, represents the complete network topology in a very straightforward manner: for any neurons n_i and n_j, $a_{ij} = 1$ if and only if n_i outputs to n_j; otherwise $a_{ij} = 0$. Their system used a GA to evolve the network topology, but the weights were initialized randomly and then learned via backpropagation. Their genetic operators included bit-flipping mutation and crossover, again, with crossover points restricted to those between each node's input-weight set. Their system solved many simple problems, such as pattern copying and XOR, but there was no clear indication that it would scale up. Since the size of their genome is quadratic in the number of neurons, N, it has serious scaling problems that the introduction of development into EANNs is meant to alleviate.

Figure 1 shows both approaches to the EANN representation problem.

Although Miller et. al. were the first to combine evolution and learning **for ANN phenotypes**, Holland's [4] original GA includes an elegant learning mechanism known as the *bucket brigade* algorithm (described in the 1975 edition of [4]. Holland's GA encodes phenotypes that denote problem-solving rules run by a simple rule-based system. The combination of Holland's GA and the RBS is known as a *classifier system* . It uses the bucket brigade to perform credit assignment along a temporal sequence of rule invocations, wherein not only rules whose application directly leadsto a solution, but also early rules that help *set up* a solution, are awarded points/credit. These points affect the prioritization of rules during problem solving, and thus, the bucket brigade embodies adaptive lifetime performance, i.e. learning.

Kitano [6] contributed the first combination of evolution and development for ANNs. His phenotype is a neural network generated from a connection table, identical in form to those of Miller et. al. However, whereas Miller et. al. use genotypes that directly correspond to these tables, Kitano uses context-free grammar rules as the basis for growing these tables, as shown in Figure 2.

Kitano employs a bit-flipping mutation operator and totally unrestricted crossover. In addition, Kitano uses a mutation rate that is specific for each newly-generated child genotype. The rate is inversely proportional to the Hamming distance between the parents. Hence, the children of similar parents are more likely to be mutated. This makes intuitive sense relative to the goal of exploration: if the parents are very different, then crossover alone with probably create unique children, but if the parents are similar, then supplementary mutation is needed find child genotypes that are not

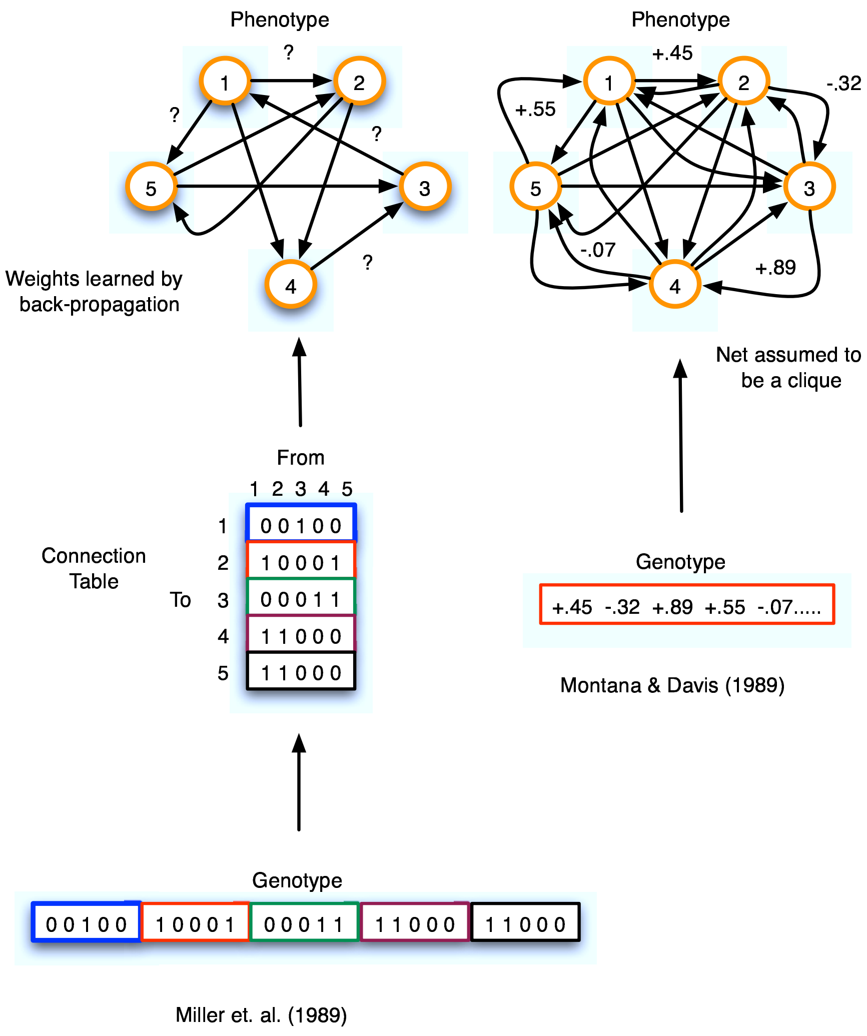


Figure 1: Comparison of genotype-phenotype conversions in the EANNs of Montana and Davis [9] and Miller et. al. [8]

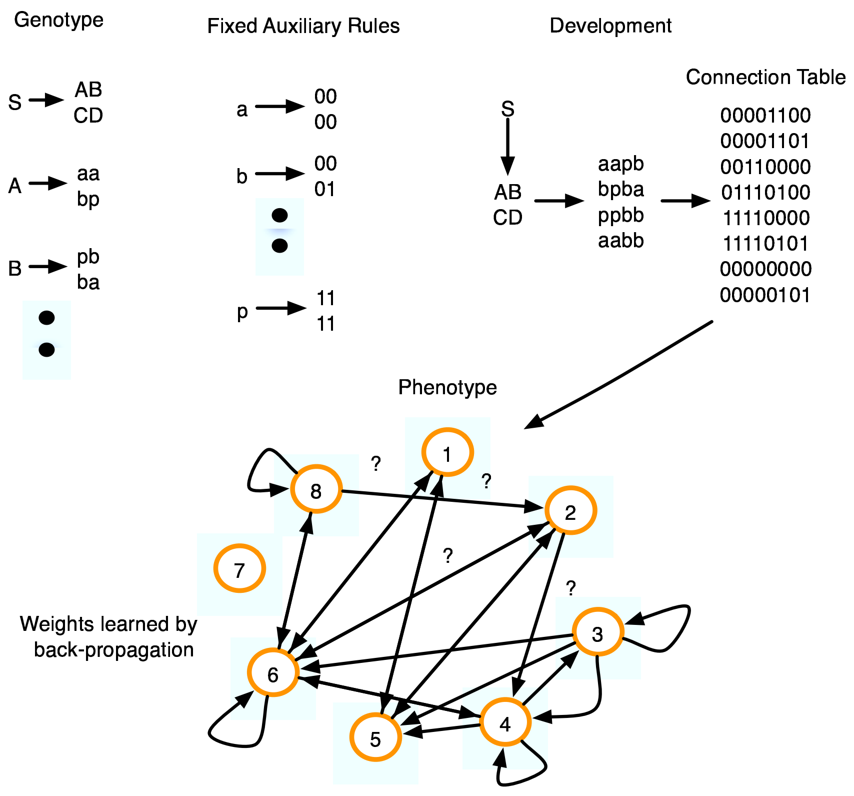


Figure 2: Kitano's [6] encoding of neural networks as context-free grammars.

clones of either parent.

Once evolved and developed, Kitano's networks receive random initial weights that are then tuned via backpropagation. Hence, Kitano was the first to design a complete POE/TRIDAP system. Also, at first glance, Kitano appears to have solved the scaling problem in that large phenotypes can be generated from short rule sets. His results are better than those of direct-coded GAs, but a) the test conditions are somewhat suspect, and b) no demonstration with large ANNs is given. Hence, Kitano's work serves as a proof of principle, but no more.

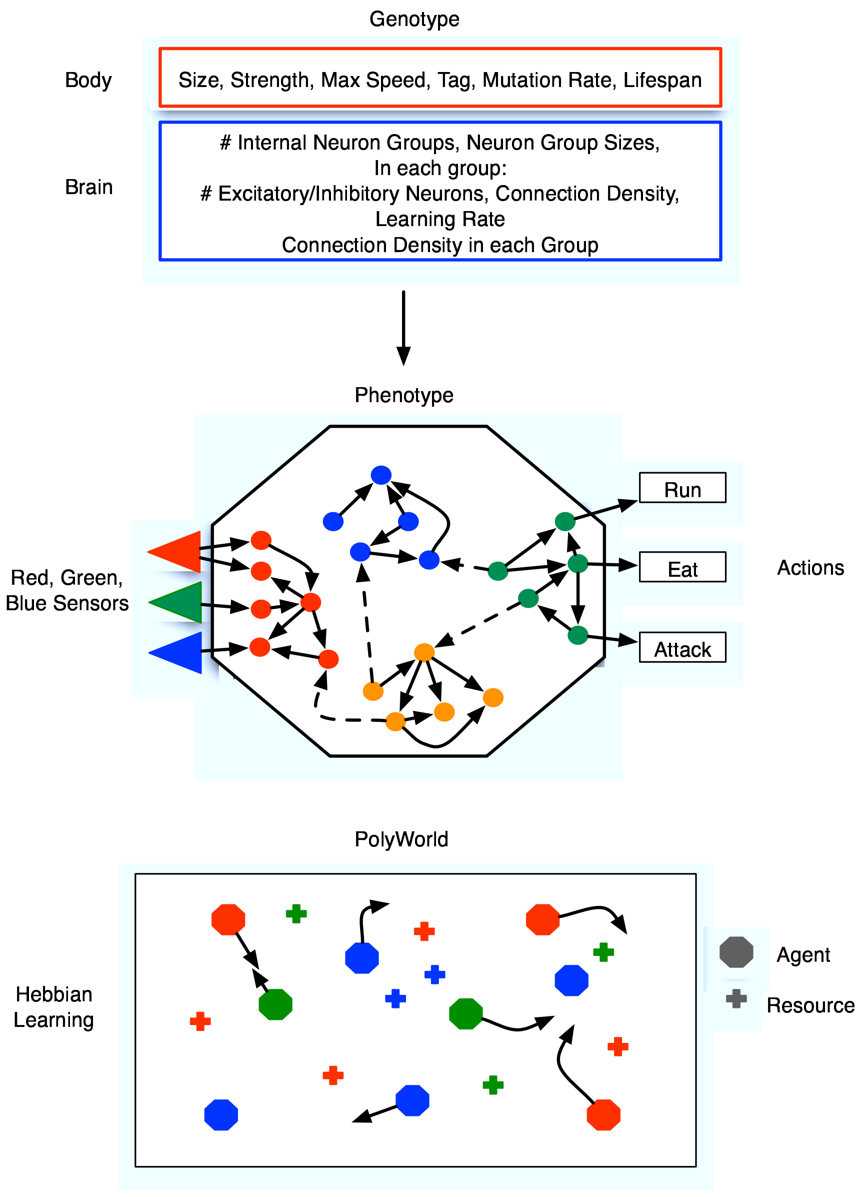

Genotype
Body
Size, Strength, Max Speed, Tag, Mutation Rate, Lifespan
Brain
# Internal Neuron Groups, Neuron Group Sizes,
In each group:
# Excitatory/Inhibitory Neurons, Connection Density,
Learning Rate
Connection Density in each Group
Phenotype
Red, Green, Blue Sensors
Run
Eat
Attack
Actions
PolyWorld
Hebbian Learning
Agent
Resource

Figure 3: Yaeger's [14] PolyWorld system.

Yaeger [14] uses the POE approach in an artificial life (ALife) environment known as PolyWorld, in which a population of agents move about, eat, mate and fight. As shown in Figure 3, each agent consists of a genotype that encodes many properties of both its body and neural-network *brain*. Hence, a significant developmental process is required to convert the genotypic parameters into complete ANNs.

ANN weights are randomly initialized and then modified by unsupervised Hebbian learning as the agent acts in the environment. The behaviors that eventually emerge in PolyWorld include edge running, flocking, and cannibalism.

Although PolyWorld uses elements of evolution, development and learning, the developmental stage is relatively minor. In effect, the genotype encodes all critical aspects of the ANN and body morphology. These specifications must still be converted into phenotypes, but this is essentially an act of assembly, not a legitimate growth process.

In game-playing, robotic and artificial life contexts, supervised learning has no practical utility, since *correct actions* cannot be provided for each sensory context by an omniscent teacher or teaching module. Although unsupervised learning is easily achieved with Hebbian weight updates, it only allows the system to learn patterns, not to associate those patterns with value judgements. Hence, although a PolyWorld agent can be born with the ability to move toward food, it has a hard time learning associations between the temporally distinct (but clearly related) events of seeing, approaching, and eating food. Hence, it cannot learn that finding food is a key prerequisite to the rewarding experience (in terms of energy gain) of eating.

These situations call for reinforced learning (RL), wherein positive or negative reinforcements at time T translate into modifications of the behaviors used at times T, T-1, T-2, etc. Ackley and Littman [1] recognized this and designed their Evolutionary Reinforcement Learning (ERL) system to investigate the interactions between evolution and learning.

Agents in ERL have brains consisting of two neural networks, one for action choice (A-ANN) and another for evaluating states (E-ANN), as shown in Figure 4. There is no development in their system: the genome encodes the initial weights for each ANN. Those of the E-ANN do not change during the agent's lifetime, while those of the A-ANN are modified via a specialized variant of backpropagation.

The networks interact as follows: The current state of their artificial life world (AL World) is input to the A-ANN, which chooses an action that (potentially) causes a change in the AL World state. This new state, as perceived by the agent's sensors, is fed into the E-ANN, which outputs an evaluation of that state. Whether positive or negative, the evaluation is used to modify the A-ANN weights so that similar actions are chosen or avoided, respectively, in similar future situations.

The AL World is a 2-dimensional grid populated with agents, plants and obstacles. Agents move about, feeding on plants or other agents; they reproduce after having accumulated significant energy, and die by running low on it.

A major impetus of the ERL project was to investigate The Baldwin Effect [2], an interesting relationship between evolution and learning (which is detailed below). ERL confirmed this effect in simulation via the following two emerging results:

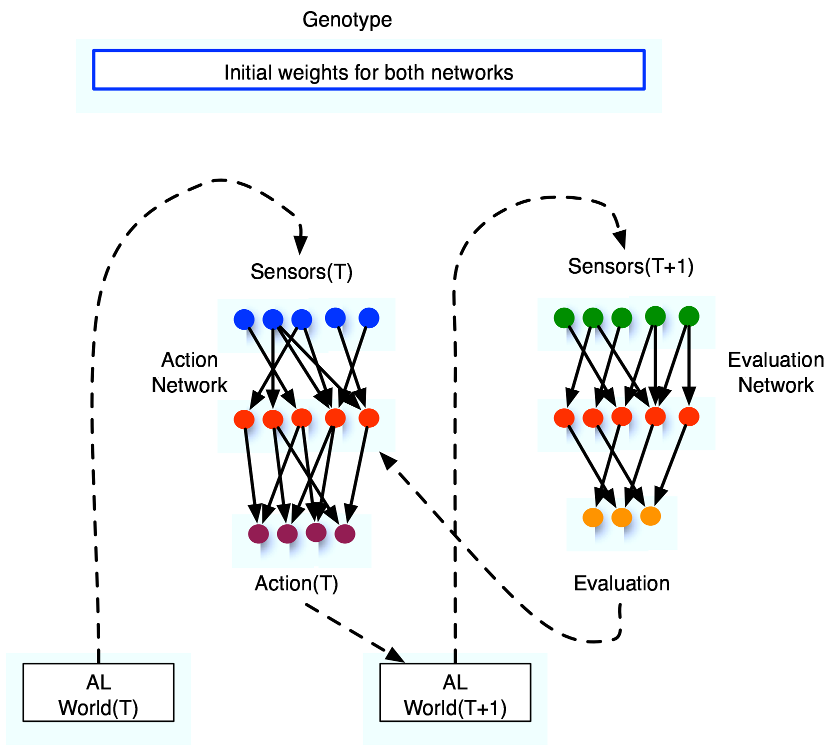


Figure 4: Ackley and Littman's [1] Evolutionary Reinforcement Learning (ERL) system for exploring the interactions between learning and evolution in an artificial life world (AL World).

1. Early in the simulations, genes coding for the weights of the E-ANNs converge to a few fixed values across the whole agent population. This low genetic variance indicates a selective pressure for learning in the A-ANNs, which depends upon accurate evaluations from E-ANNs. This mirrors stage I of The Baldwin Effect, where learning is important and thus, learning rates are high.

2. Later in the simulation, these same E-ANN genes diverge via *genetic drift.* This high variance indicates low selective pressure for good weights in the E-ANNs, and hence, no strong need for learning in the A-ANNs, (since the A-ANNs evolve to have innately good weights). This embodies stage II of The Baldwin Effect, wherein *natural-born talents* (who require little learning) arise.

4 Biological Theories of Evolution and Learning

One of the first proposals that learning could accelerate evolution was Jean-Baptiste Lamarck's (1744-1829) *inheritance of acquired characteristics*, wherein physical and mental changes incurred during one's lifetime could be passed on directly to offspring. Contemporary knowledge of the germ-soma distinction permits a recasting of Lamarckism in modern Neo-Darwinian terms, as depicted in Figure 5. Thus, the theory entails a reverse transcription of the modified phenotype back into the genotype, a process that is fully realizable and often useful in evolutionary algorithms, but biologically unrealistic except in a few rare cases.

In 1896, James Baldwin postulated an indirect mechanism for the eventual inheritance of acquired characteristics [2]. This *Baldwin Effect* involves two stages. In phase I (Figure 6), assume a set of genotypes spread uniformly about a sub-optimal region, D1, of the fitness landscape. If phenotypes have plasticity, then each can essentially perform local search in the fitness landscape (as shown by the circles with horizontal arrows), and a rough estimate of phenotypic fitness will be the time-averaged landscape locations of the phenotype. Clearly, those phenotypes lying near the base of the optimal peak will have better opportunities to learn their way to higher fitness. Hence, they will have a selective advantage, and the population distribution will move from D1 to D2. Basically, learning smoothes the fitness landscape and enhances selective pressure such that the population moves toward the optimal phenotype, denoted by P* on the phenotype axis.

To move, and not merely redistribute, the genotype pool, evolution relies on genetic operators (mutation, crossover, inversion, etc.). If the genotype and phenotype space are well correlated [7], then genetics can initiate the emergence of innately optimal phenotypes, *natural born P*s*, and, in general, lead to a flattening of distribution D2 into D3 (Figure 7). Additionally, if learning has a cost, as it normally does [7], then the *P* learners* will pay it but the natural-born P*s will not, thus giving the latter a selective advantage and moving the population distribution from D3 to D4, where the learned phenotype, P*, becomes fully innate.

Thus, in the Baldwin Effect, learning accelerates evolution; and then, if the fitness landscape is static, evolution obviates learning via genetic assimilation.

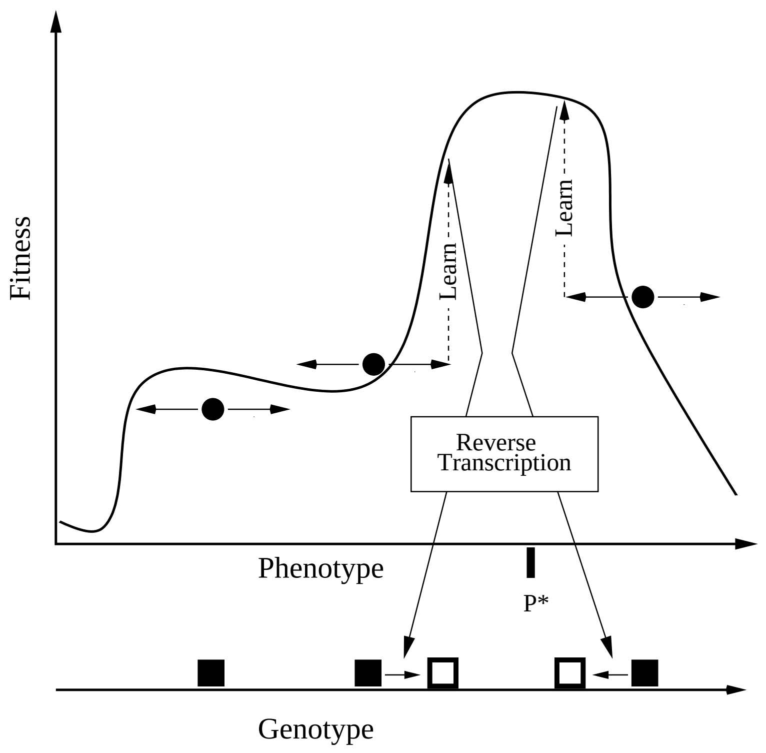


Figure 5: Neo-Darwinian Interpretation of Lamarckian Inheritance of Acquired Characteristics: During their lifetimes, phenotypes (circles) can improve and thus climb the fitness landscape away from their innate position and toward the optimal phenotype, P*. Any phenotypic improvements are then reverse transcribed into the genome (changing from filled to open square) prior to reproduction.

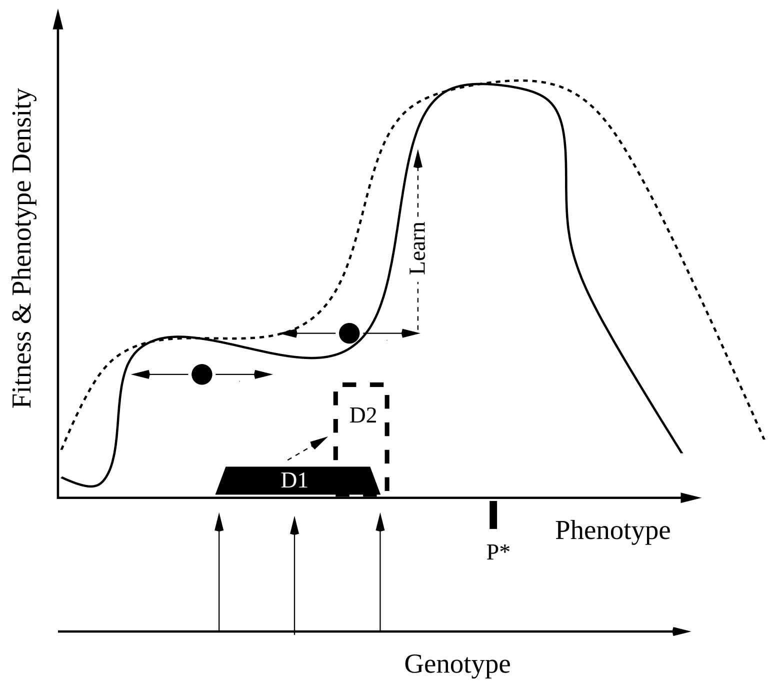


Figure 6: The Baldwin Effect Phase I: All phenotypes have the ability to learn, but only those near the base of the peak can achieve fitness increases over the innate value. This selective advantage moves the genotype/phenotype distribution from D1 to D2, hence closer to the optimal phenotype, P*. As depicted by the dotted curve, learning effectively smoothes the fitness landscape.

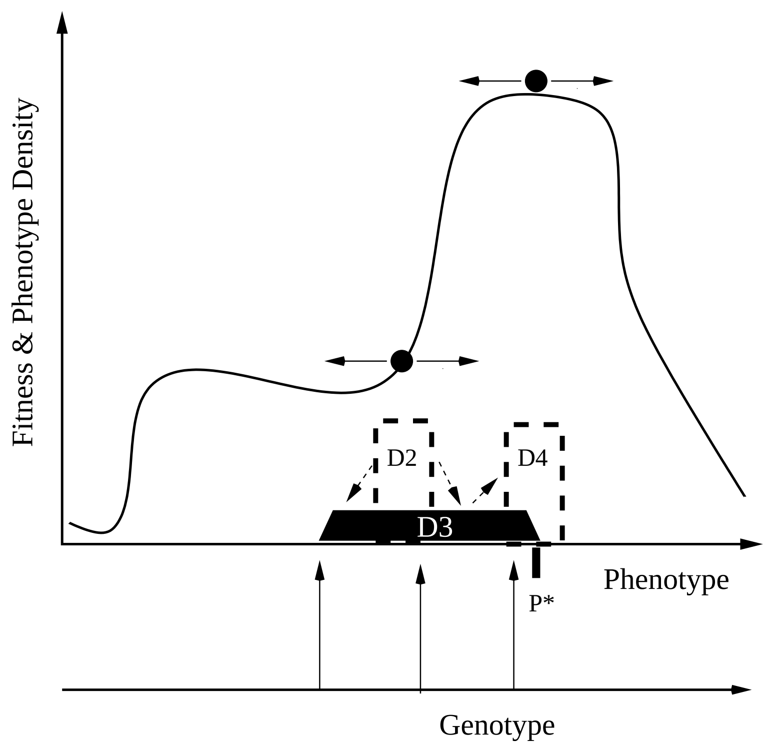


Figure 7: The Baldwin Effect Phase II: Genetic operations spread the population distribution from D2 to D3, producing individuals with hard-wired optimal phenotypes, P*. Due to the cost of learning, these *natural-born P*s* have a selective advantage over the *learned P*s*, and the distribution moves from D3 to D4.

5 The Competing Conventions Problem

As described in [13] and originally recognized by Radcliffe [11], the symmetry of neural network solutions poses a permutation problem for evolutionary ANNs. Basically, the same functional unit can appear in different locations of different neural networks, such that crossover of these networks produces functional redundancy (and the concomitant loss of other functions). As a simple example, consider two networks for performing exclusive or (XOR), as shown in Figure 8.

This is a permutation problem, since N behaviors can be distributed N! ways across N internal nodes of an ANN. Crossover among any two ANNs that do not employ the exact same permutation can potentially lead to functional redundancy and loss. As described by Whitley [13], prior to 1995, several solutions had been proposed for this problem, but none were convincing enough to make evolving ANNs a more efficient approach than backpropagation for the general case of supervised learning in standard feedforward ANNs. In addition, some researchers even concluded that competing conventions was not a serious problem at all.

However, in 1997, Moriarty and Miikkulainen [10] devised the SANE system, which effectively evolves individual neurons, thus circumventing the Competing Conventions problem.

In SANE, the ANN topology is predetermined, with a fixed number of input, hidden and output nodes. Networks are strictly feedforward, but the three layers are not necessarily fully connected to one another. Each EA genotype encodes functional parameters for one hidden-layer neuron. For any hidden node, H, this specification consists of pairs (index, weight), where the index selects an input or output node, and the weight codes the value for the arc between that node and H. The top of Figure 9 illustrates this encoding.

Individuals specifications are then evaluated in terms of their ability to cooperate with other specifications to form complete ANNs. Essentially, each individual participates in many random groupings with other individuals, with the fitness of each individual being the sum fitness of the best 5 groups in which it participates. The bottom of Figure 9 summarizes this process.

Extensive testing of SANE in domains such as game-tree search and robot control verifies that it:

- finds good solutions faster than standard whole-network evolutionary approaches,
- maintains population diversity over the entire simulation (since individuals cooperate, not compete, with other individuals),
- quickly adapts to changing environments.

SANE skirts around the Competing Conventions problem by never explicitly linking neurons together in the genome. Although several similarly-functioning neurons may exist in the population, functional heterogeneity is maintained naturally, since a) collections of **diverse** cooperating neurons are needed to solve problems, and b) all neurons cooperating in successful ANNs are rewarded with higher fitness.

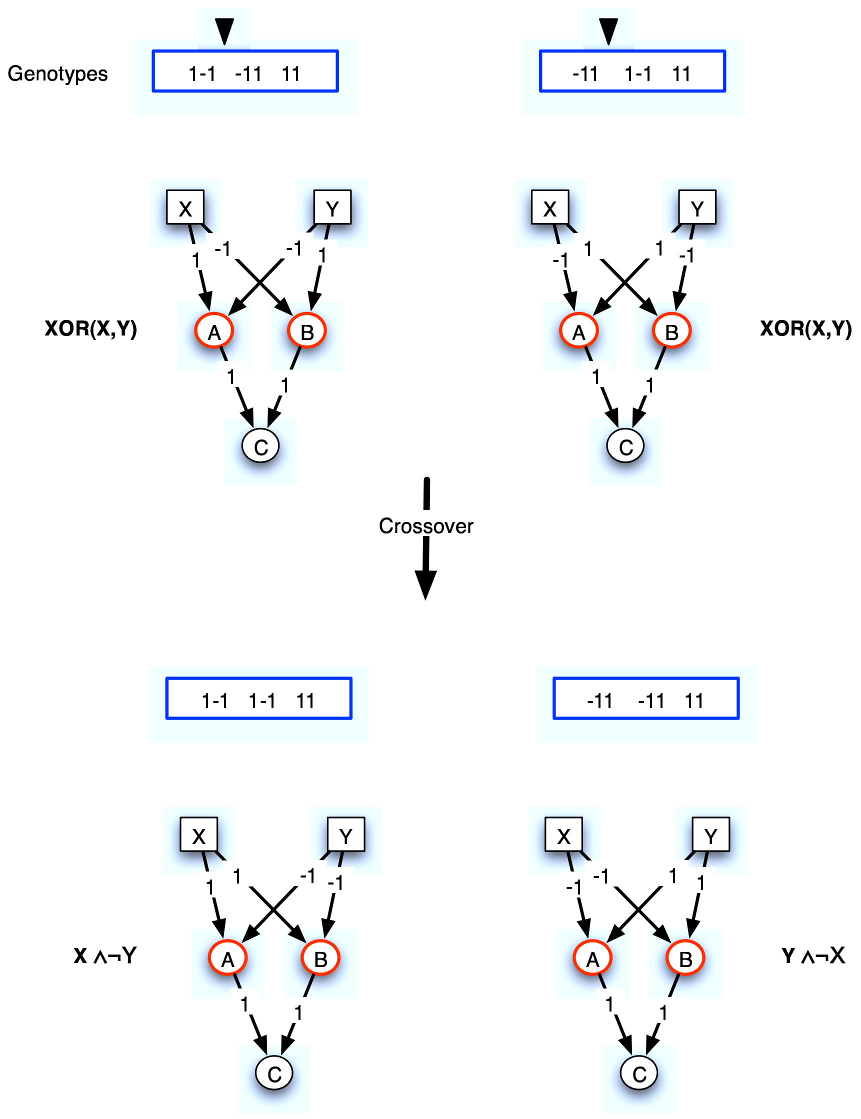
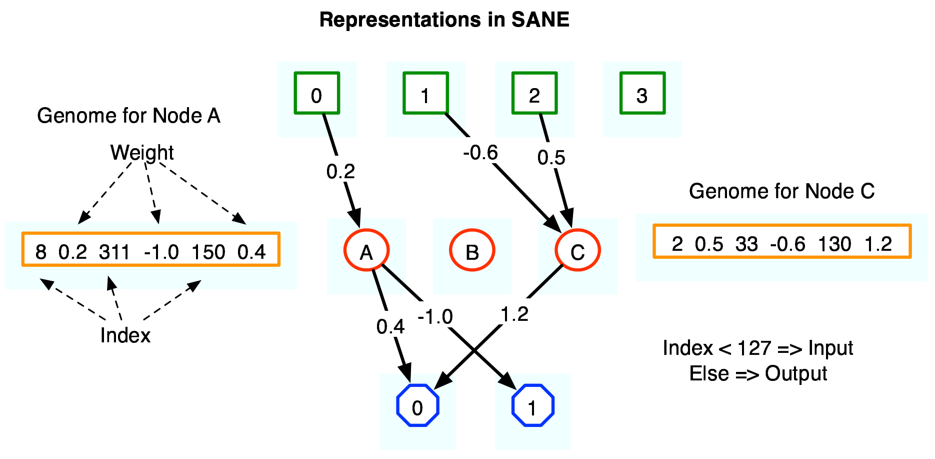


Figure 8: Illustration of the Competing Conventions Problem. Both upper ANNs compute XOR, but with internal nodes A and B performing opposite roles in each ANN. After crossover, the children each get a pair of functionally equivalent nodes and no longer compute XOR. Dark triangles above the genotypes denote crossover points.



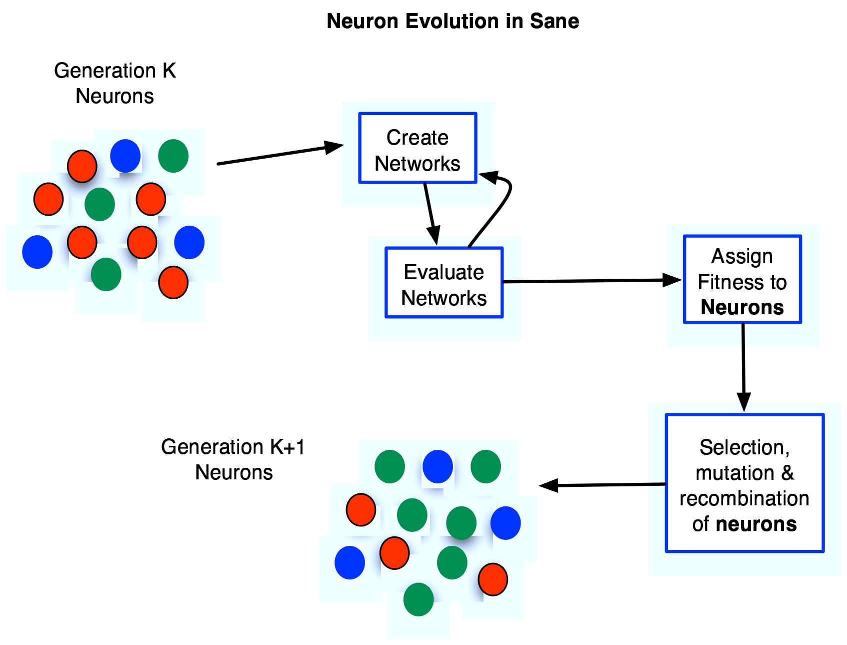


Figure 9: The SANE system [10] for designing neural networks by evolving populations of individual hidden-layer neurons. (Top) The genotypic encoding of (index, weight) pairs for the hidden-layer neurons. (Bottom) SANE's evolutionary process, wherein a neuron's fitness is the sum of the fitnesses of the best 5 ANNs in which it participates.

The heart of the Competing Conventions problem is the inability to detect functional similarities among subcomponents, thus leading to redundancies and omissions during crossover. A similar problem arises when chromosomes are allowed to change size during evolution. This is a desirable property in terms of complexification: genomes begin small/short and gradually grow as evolution finds good regions of the smaller search spaces. This is exactly how nature has operated over the last billion years or so. Genomes have gradually become more complicated via the combination of duplication and differentiation.

For example, consider a gene G that is tied to some trait T. Assume that a copy of G, G*, appears on the chromosome, where G* also links to T. The addition of G* presumably has no effect upon the phenotype other than to give extra support for the development of T. Now, since G* is redundant, it is free to mutate to other *neutral* genes (i.e., those that do not affect fitness). Hence, as long as G* does not code for lethal or otherwise deleterious traits, it is free to explore gene space. Eventually, it may mutate to a fitness-enhancing trait, T*, thus moving the genome to a more complex (and fitter) point in the search space. Clearly, the duplication has provided the genome with a certain *exploratory freedom*, since a) trait T is still covered by G, and b) G* initially codes for something useful, thus giving it a good starting point for further investigation.

EANNs can benefit from duplication and differentiation as well, thus supporting complexification. However, competing conventions can intervene, as shown in Figure 10. Here, an initial gene triple, ABC, duplicates in two different ways, to produce ABCDE and ABCFGH. In the figure, shape indicates the phenotypic trait for which the gene codes; shape changes during the differentiation phase.

When crossing over the unequal-length genomes, alignment becomes a problem. The variant on the left produces children with either redundant or omitted C genes. The child ABFGH has no C and thus cannot produce the octagon trait, while ABCCDE has extra octagons but no trapezoid.

On the right, each child gets a single copy of ABC, but neither gets all three of the new traits: circle, trapezoid and square. Instead, each gets a duplicate of one trait.

Thus, alignment difficulties cause two problems: redundant genes and omission of phenotypic traits. Thus, although two individuals may acquire distinct and useful traits, their mating is not guaranteed to produce children with all of the good traits. Hence, independently-evolved building blocks will not necessarily unite to produce superior phenotypes.

To solve this problem, Stanley and Miikkulainen [5] use an abstraction of the biological process of synapsis, wherein identical genes align during recombination. In their NEAT system, each gene includes a historical marker that denotes its relative time point of origin. As shown in Figure 11, this allows genomes to align properly, with identical genes always appearing in corresponding spots. Hence, crossover itself a) never causes gene duplication, and b) stands a better chance of combining phenotypic building blocks. The latter fact stems from the simple observation that two genes coding for different traits will never align, and thus never be prevented from appearing in the same child. They may not always combine, but they will have the following probability of doing so:

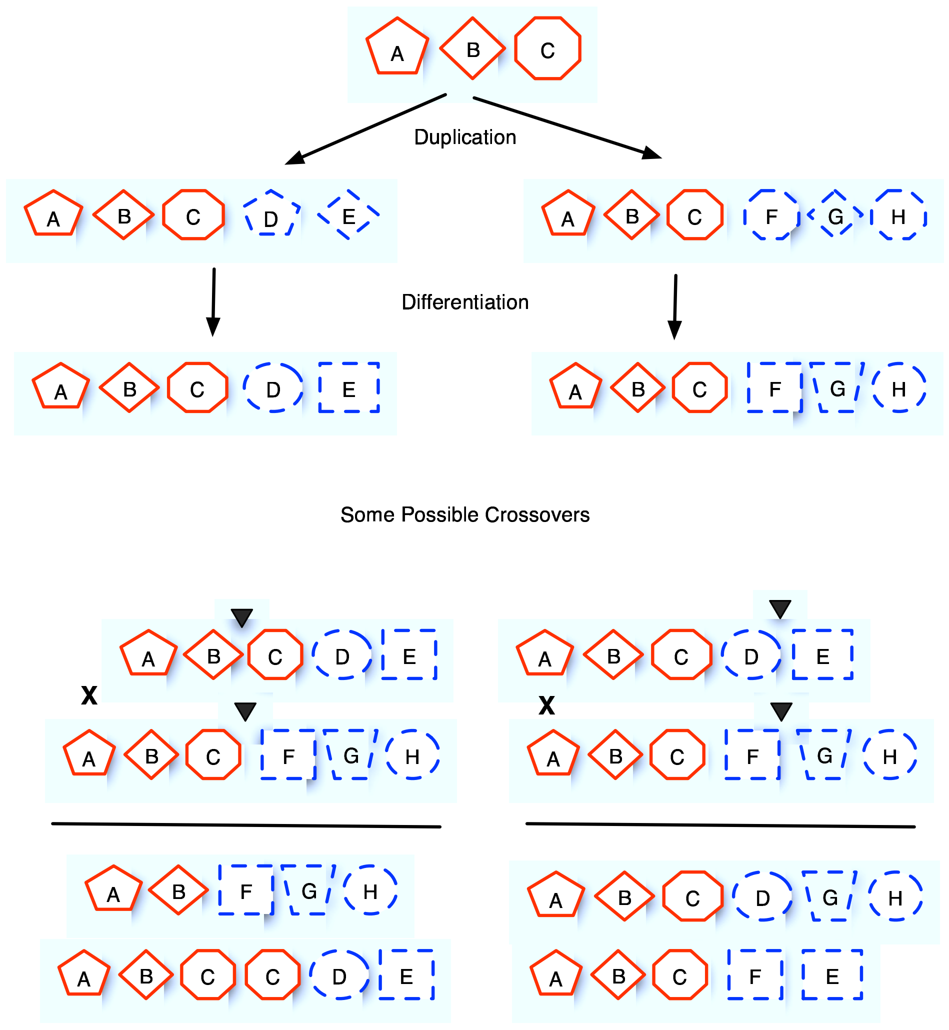


Figure 10: Competing Conventions problems caused by duplication, differentiation and the resulting unequal-length chromosomes. Shape denotes the corresponding phenotypic trait, while line type represents the original (solid) or duplicated (dashed) genes. Solid inverted triangles are crossover points.

$$p_{cross} \times \frac{\mid HM(X) - HM(Y) \mid}{N - 1} \tag{1}$$

where p_{cross} is the probability of single-point crossover, N is the total number of genes in the population and hence the length of the aligned, gap-filled genomes during crossover, and HM(X) and HM(Y) are the historical markings of genes X and Y. Basically, the closer that X and Y are in time of origin, the closer they will appear on the aligned genomes, and thus the lower the probability that the crossover point will be chosen between them. And of course, they can only be combined on a child chromosome if the crossover point does split their locations.

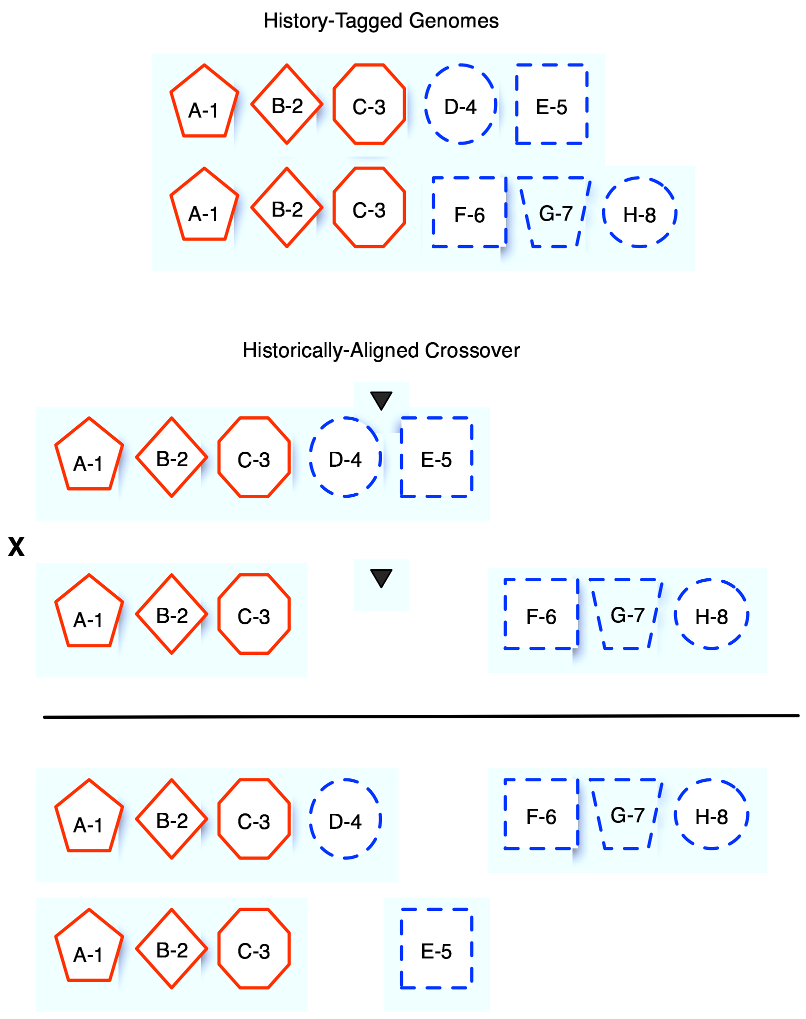


Figure 11: Avoiding Competing Conventions with historical markings, as used in Stanley and Miikkulainen's NEAT system [5]. The relative time of origin of each gene appears as a hyphenated addition to its identifier. Crossover aligns genomes according to historical markings, thus preventing gene redundancy and broadening phenotypic coverage in child genomes. However, duplication of phenotypic traits can still occur, as shown by D-4 and H-8 in the upper child genome.

The actual genotype representation in NEAT codes for the number and type of network nodes, along with their connections and final weights; no learning is involved. Networks are initialized to have 3 layers: input, hidden and output, although the hidden layer can effectively have many topologies, some of which give it a multi-layered look. A typical genotype and corresponding phenotype appear in Figure 12. Genomes are mutated by either adding a new node (and a connection into and out of it) or adding a new connection between existing nodes. Crossover uses historical markings, with each node and connection gene having its own history stamp, as depicted in Figure 11.

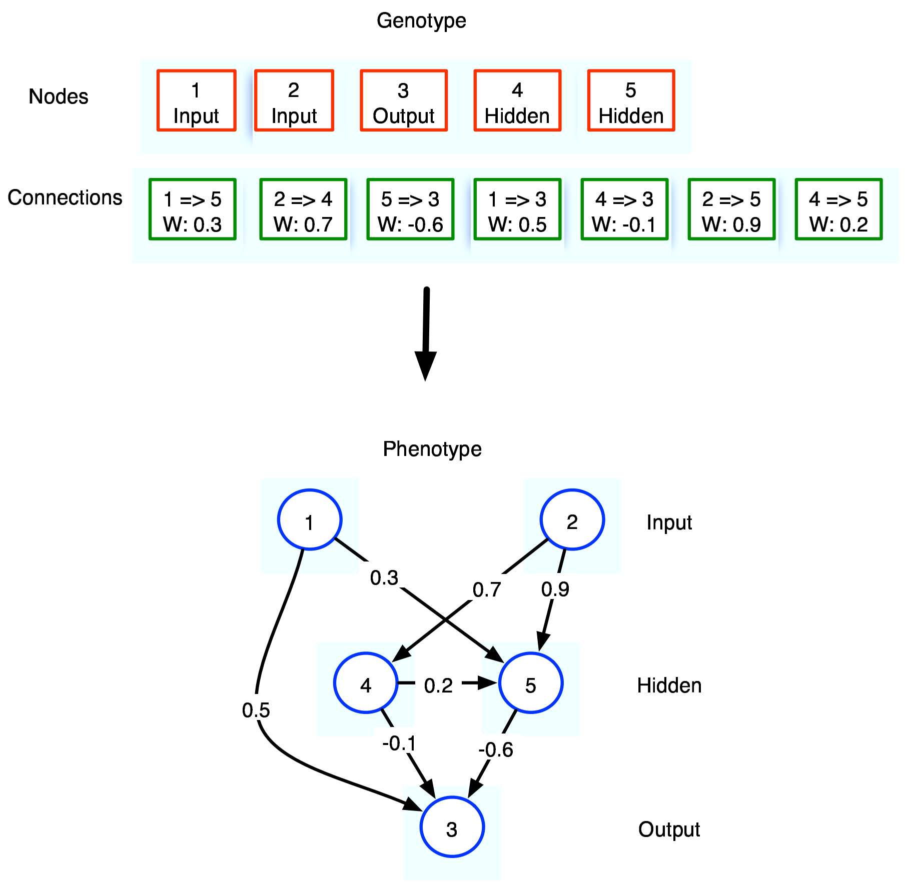


Figure 12: A typical genotype and corresponding phenotype in NEAT. Each rectangle denotes a gene. Historical markings on both node and connection genes are not included.

5.1 Speciation for Complexification

In evolutionary systems, innovative phenotypes with useful new traits often arise. However, unless those traits integrate nicely with pre-existing traits to form a *better* organism, the new phenotype may lose in competition with its contemporaries. Essentially, evolution rarely gives new designs an opportunity to *work out the bugs*. The sieve of natural selection has no compassion.

Evolutionary algorithms are equally ruthless, but they can be supplemented with explicit niching mechanisms such that new designs have a *fighting chance* against established solutions. For example, fitness sharing [3] divides an individual's raw fitness by the number of individuals that resemble it, where resemblance can entail similar fitness values or similar genotypes. Clearly, the latter condition seems more reasonable, since two like-fitness individuals need not be similar solutions. Fitness sharing prevents a population from converging, since as more individuals become similar, they must share fitness among a larger and larger group, thus reducing each agent's worth. And this allows new, unique, solutions to get a foothold in the population, since a) they have some promising properties, and b) they need not share fitness with a large group.

Unfortunately, judging similarity among graph-like structures such as neural networks is an NP-complete problem, so proper fitness sharing would appear intractable. However, the NEAT developers solved the problem by using historical markings, which enable a simple similarity check among pairs of individuals: compare their marking lists. NEAT exploits this similarity metric and fitness sharing to integrate, not filter, innovation. In this way, neural networks can gradually complexify over the evolutionary generations.

Stanley and Miikkulainen [5] view gradual complexification as the only feasible route to complexity. To begin with large genomes creates an intractable search space. By beginning small, useful general solutions can be found; they then improve by elaboration as new genes arise to code for special accessory traits. In the end of an evolutionary run, the final genome may be large, but the entire space that it represents never needs to be searched. Only subspaces of proven utility are investigated.

Gradual complexification in NEAT leads to impressive performance results. It defeats four other popular approaches, including SANE [10] and Cellular Encoding [**?**], on pole-balancing problems. In addition, NEAT is the adaptive mechanism in an intriguing new video game, NERO, in which teams of warriors are trained via an interactive evolutionary algorithm in which a) the human user adjusts fitness parameters according to the desired type of fighting unit, and b) evolution modifies the neural-network controllers of the fighters.

In summary, NEAT supports gradual complexification of neural networks via duplication and differentiation of genetic material. It does so by including a simple, yet powerful, piece of information in each gene: a historical marking. This concept can supplement many EAs, whether they evolve neural networks or other structures.

6 Evolving Complex Neural Networks

Our central philosophy is that ANNs are the optimal representational medium for truly adaptive, human-like behavior, but their topological layouts and the properties of individual neurons and synapses (of which there are often tens or hundreds of thousands) cannot be pre-determined by human engineers in anything more than an ad-hoc manner. The search space for complete ANN designs that can achieve sophisticated behavior is simply ominous.

How, then, can artificial systems design truly complex ANNs for producing intricate human-like behaviors? First of all, the meaning of *complex* needs careful attention. If size is a sufficient measure of complexity, then straightforward developmental routines easily allow a small grammar to generate a large ANN. In that case, development would be the obvious key to scaling up EANNs.

Unfortunately, complexity involves much more than size. Many large structures, such as classic office buildings, involve multiple copies of basic patterns for floors, rooms, windows, etc. A complete building plan for a 100-story office tower may only include details for a few of the individual floors, such as those near the bottom and top, those where elevator transfers are necessary, etc. while the remaining floors are just instantiations of a standard blueprint. The same is true of many large circuits, which contain numerous copies of basic off-the-shelf components, often arranged in repeated, stereotypical topologies. In short, the size of the *recipe* for producing the building or circuit is much smaller than that of a detailed non-generative description of the finished artifact.

Truly complex structures are those whose descriptions cannot be compressed into concise generative routines. Any generator would be as long (or longer than) a non-generative description.

It is quite possible that the brain of a well-trained animal exhibits this form of incompressible detail. Even a restricted subset of the brain, say those areas needed to recognize and pursue prey, probably contain several million neurons. Finding the proper weights for a collection of a million interconnected artificial neurons defies current technology. We now consider the prospects for each of the 7 subsets of the 3 adaptive mechanisms to eventually handle problems of this size.

First, evolution alone has little chance of handling the problem. A network of a million nodes might have nearly a billion connections. Genomes of that size are unheard of in the EA community and will not be for the foreseeable future.

Similarly, development alone could hardly generate a billion-link network unless it contained huge amounts of symmetry/redundancy. Such large-scale repetitive structure would probably not suffice to handle the intricacies of realistic animal behavior. Without the fine-tuning capabilities of experience-based synaptic modification, development can only scale up to very regular structures.

In isolation, ANN learning techniques begin with a random set of weights and must tune them to handle the current task. Techniques such as backpropagation must often process training sets thousands of times, even when training very small networks. Whereas animals can learn from a single experience, ANNs using backpropagation are embarassingly slow learners. The inability of gradient-descent methods to efficiently handle recurrent networks is also a big problem, since all complex circuitry in animal brains is highly recurrent. Of course, ANNs could be trained by more Hebbian means, but these techniques have shown only moderate success to date.

ANNs that only learn must normally do so with a fixed, pre-defined topology. Standard 3-layered (input, hidden, output) topologies work well on a wide variety of mapping problems, with the main question being the number of hidden nodes. Fully connecting each layer to its successor layer also works well in the general case, with irrelevant links simply attaining weights close to zero. However, these extra links increase the training time of the network. Heterogeneous topologies that burden learning with only relevant connections are desirable, but these are difficult to design by hand.

The combination of evolution and learning has some promise, since evolution can save ANNs from costly training procedures, but, unfortunately, at the expense of evaluating hundreds or thousands of different ANNs. Still, evolution has no problem with recurrent connections, so one division of labor might be to allow genomes to encode fixed weights for recurrent links, while standard feed-forward connections could be learned. To more closely mimic the brain, the learning might be a combination of unsupervised Hebbian modification (to learn patterns) and reinforcement learning (to learn associations between actions and rewards).

Combining development and learning has promise with respect to both size and intricacy aspects of complexity. To wit, the former allows us to create large, structured networks, while the latter enables situation-dependent weight tuning. However, the two in isolation present a bit of a dilemma:

1. If an engineer already knows the desired topological structure of an ANN that will serve as the basis for learning, then the design of a developmental process to *grow* that network is little more than an interesting exercise, unless, of course, the hand-crafting of the topology is tedious and requires automation.
2. If the topology is not known ahead of time, then the system will need to **search** in the component and parameter space of the developmental procedure in order to grow many different networks, which are then tested for their ability to synaptically tune to the problem.

Hence, it makes little sense to have development unless it is accompanied by a search technique. Unfortunately, this type of search problem is not nicely constrained: it involves finding primitive sets for growing useful global structures. Many traditional search techniques would bog down in their attempts to enumerate and evaluate all possible combinations of primitives, but evolutionary algorithms have no problem with these more untraditional search spaces. Anyone who has ever written context-free grammars or rule sets for expert systems knows the difficulties of designing primitives for growing complex structures or achieving complex behaviors. The work requires considerable creativity and trial-and-error. Hence, for complex problems, development may actually require evolution.

EvoDevo systems, i.e., those that combine evolution and development, are very popular, since they do allow small genotypes to produce large (structured) phenotypes, and evolution can effectively search for a useful developmental strategy in the restricted genotype space. However, these systems fail to capture problem-relevant intricacies. Granted, a relatively simple signalling system during development, using only a few overlapping chemical gradients, can achieve a reasonable degree of heterogeneity in the phenotype. This may even enable the formation of neural structures such as topological maps [12]. However, development simply cannot anticipate all the critical neural associations that organisms need to form in order to behave properly in their environments.

Nature versus nurture debates always seem to come to similar conclusions: the emergence of complex behavior requires both. Hence, the brains of complex organisms cannot be pre-wired to handle all of life's complexities; learning is necessary.

Thus, the biological and engineering realities indicate that the achievement of sophisticated intelligence in an artificial neural substrate may require all three adaptive mechanisms. In short, scaling requires development, which requires evolutionary search; and both need help from learning to tailor a large set of connections to the environment and problem at hand. Although many contemporary POE/TRIDAP systems are primarily proofs of principle, this basic combination appears essential for the scaling of connectionist AI.

References

[1] D. H. Ackley and M. L. Littman, *Interactions between learning and evolution*, in Artificial Life II, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds., Reading, Massachusetts, 1992, Addison-Wesley, pp. 487–509.

[2] J. M. Baldwin, *A new factor in evolution*, The American Naturalist, 30 (1896), pp. 441–451.

[3] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley Longman, Reading, Massachusetts, 1989.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA, 2 ed., 1992.

[5] Kenneth and R. Miikkulainen, *Evolving neural networks through augmenting topologies*, Evolutionary Computation, 10 (2002), pp. 99–127.

[6] H. Kitano, *Designing neural networks using genetic algorithms with graph generation system*, Complex Systems, 4 (1990), pp. 461–476.

[7] G. Mayley, *Landscapes, learning costs and genetic assimilation*, Evolutionary Computation, 4 (1996).

[8] G. F. Miller, P. M. Todd, and S. U. Hedge, *Designing neural networks using genetic algorithms*, in Proc. of the Third Int. Conf. on Genetic Algorithms, San Francisco, CA, 1989, Morgan Kaufmann, pp. 379–384.

[9] D. J. Montana and L. D. Davis, *Training feedforward networks using genetic algorithms*, Proceedings the Eleventh International Joint Conference on Artificial Intelligence, (1989), pp. 762–767.

[10] D. E. Moriarty and R. Miikkulainen, *Forming neural networks through efficient and adaptive coevolution*, Evolutionary Computation, 5 (1997), pp. 373–399.

[11] N. Radcliffe, *Genetic neural networks on MIMD computers*, PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1990.

[12] N. Swindale, *The development of topography in the visual cortex: A review of models*, Network: Computation in Neural Systems, 7 (1996), pp. 161–247.

[13] D. Whitley, *Genetic algorithms and neural networks*, in Genetic Algorithms in Engineering and Computer Science, J. Periaux and G. Winter, eds., John Wiley and Sons, 1975.

[14] L. Yaeger, *Computational genetics, physiology, metabolism, neural systems, learning, vision and behavior or polyworld: Life in a new context*, in Artificial Life III, Proceedings Volume XVII, C. G. Langton, ed., Reading, Massachusetts, 1994, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, pp. 263–298.