4 Indirect encodings

When neural networks are encoded directly, the elements in the genetic representation correspond one-to-one to elements in the neural network. Indirect encodings, on the other hand, utilize a mechanism that allows expanding a compact genetic encoding into much larger and more complex neural networks. Several such approaches are reviewed in this chapter. The first three represent different levels of abstraction of indirect encoding in biology, i.e. development through cellular growth, grammatical encoding, and learning. Next, indirect encoding through hypernetworks is reviewed, where one network indirectly encodes the design of another. Finally, we're looking at dynamic indirect encodings through self-attention mechanism.

4.1 Why Indirect Encodings?

Biological organisms in nature all develop from a single starting cell. Through local cell interactions and growth over time, an initially unassuming mass of cells eventually transforms into a complex and sophisticated structure with specialized cells and intricate connections. This process of growth and development, known as morphogenesis, is a fundamental aspect of biology that underlies the formation of all living organisms. In the case of the human brain, this process is particularly remarkable, as it gives rise to the most complex and sophisticated structure known to science, with billions of neurons and trillions of connections.

The human brain exhibits a complex network of interconnected modules, which form the basis of intelligence. How this intricate structure is encoded in our genetic code, consisting of approximately 24,000 genes or 3 billion base pairs (International Human Genome Sequencing Consortium 2004), is a fascinating question that we're still struggling to completely answer. Although learning plays a crucial role, much of this information is already encoded in the genome.

To achieve this remarkable feat, regularity is necessary, which involves reusing structural motifs to enable compression and compactness of the genome. Interestingly, regularity also provides computational advantages to neural structures, as seen in the success of convolution in deep learning. Convolution, a pattern of connectivity that uses the same feature detector at multiple locations in a layer, has proven to be a powerful solution for capturing translation-invariant features in deep learning architectures. Instead of designing such patterns and others by hand and ultimately being limited by a human designer, ideally, our

neuroevolutionary algorithms would identify these powerful regularities in an automated way. This is the idea behind *indirect encodings* in neuroevolution.

Before we go into more details about indirect encodings, let's revisit the NEAT algorithm from the previous chapter. Is NEAT an indirect encoding? It is in fact an example of a *direct encoding*. There is no compression involved or any type of reuse of information, resulting in a one-to-one mapping between the parameters of a NEAT genotype (the description of the nodes that exist in the network and how they are connected to each other) and those of the neural network phenotype. In other words, for every connection in the neural network, there exists a parameter in the underlying genotype. As we have seen, NEAT works well for many problems but because it is a direct encoding it has the drawback that every subpart of the solution needs to be reinvented separately by evolution instead of allowing the genome to reuse it. It is thus not very surprising that NEAT by itself has been mostly employed for tasks that require smaller neural networks, in some instances up to a few thousand connections, but certainly less than the millions or more parameters employed in current deep reinforcement learning approaches.

Let's look at an example of what this means for a particular problem. Imagine you want to evolve a controller for a quadrupedal robot. This task likely would benefit from an approach that takes into account the underlying task patterns and symmetries; in other words, knowing how to control one leg is likely helpful in controlling the rest. A tried and tested approach for resolving such a problem using an evolutionary algorithm is to assist it in recognizing patterns and symmetries. This method involves manually breaking down the problem into smaller components, such as designing the controller for one leg of a quadruped and then duplicating it for each leg, with slight variations in phase. By doing this, the algorithm is encouraged to adopt a modular approach and employ a single encoding for multiple modules. However, it would be ideal if the algorithm would be able to automatically take advantage of the symmetry and regularities of the tasks without an engineer manually having to decompose the problem. While it is easy to see how the problem could be decomposed into sub-solutions for a quadrupedal walker, it is not always as straightforward. The idea behind indirect encodings is to address this issue through representations that have the ability to capture and express regularities such as symmetries and repetition in the phenotypic structures automatically.

Indirect encodings draw inspiration from the compression of DNA in natural systems and have a long research history stretching back several decades, including early experiments in pattern formation. Researchers have explored the use of evolvable encodings for a diverse range of structures ranging from simple blobs of artificial cells to complex robot morphologies and neural networks (Stanley and Miikkulainen 2003; Gruau 1994; Bongard, Pfeifer, et al. 2001; Hornby and Pollack 2002; Miller and Turner 2015; Doursat, Sayama, and Michel 2013).

In evolutionary computation, the process of how the genotype is translated into the phenotype, which entails all the observable characteristics of an organism, is usually called the genotype-to-phenotype mapping. In nature this mapping is achieved through the process of development. Thus, one way to take advantage of indirect encodings is to mimic development in biology (Miikkulainen and Forrest 2021). There are three main

approaches: modeling cellular growth processes, abstracting development into a grammatical rewrite systems, and combining evolution synergetically with learning. These are the topics discussed in the next section.

The two sections after that review fundamentally different mechanisms of indirect encoding. The first one is hypernetworks, in which one neural networks encodes the weights of another neural network. While developmental systems are suitable for modeling natural structures and self-similar patterns, neural networks give us more flexibility in generating diverse and rich patterns. They can not only capture regularities such as symmetry and repetition but also more complex patterns such as repetition with variation. Following, we look at how hypernetworks can be extended to serve as dynamic encodings, in which the generated weight pattern can be made input dependent. This type of dynamic indirect encoding is closely related to the idea of self-attention. How they can be the basis for an indirect encoding is the focus of the last section in this chapter.

4.2 Developmental Processes

As discussed in Section 14.4, development is a fundamental way in biology to construct complex solutions. Instead of specifying the final solution directly, evolution specifies a developmental process, i.e. the initial structure and a mechanism for building a full solution through intrinsic growth or through interactive adaptation to the environment. Such mechanisms can be harnessed in artificial systems as well. Emulating biology, many different developmental mechanisms can be used to establish artificial embryogeny (Stanley and Miikkulainen 2003), i.e. a biologically inspired way to take advantage of indirect encodings. One way is to emulate cell-chemistry mechanisms such as cellular growth and genetic regulation. Another is to abstract development into grammatical rewrite steps. A third is to take advantage of learning, either individually or through population culture. These ideas will be reviewed in the subsections below.

4.2.1 Cell-Chemistry Approaches

Understanding the fundamental characteristics of natural patterns has been an important motivation for developmental systems. In seminal work in 1952, Alan Turing proposed a system based on diffusing chemicals, successfully simulating patterns reminiscent of those found on seashells, feathers in birds, and fur in mammals (Turing 1952). At the other end of the spectrum, Aristid Lindenmayer in 1968 proposed high-level grammatical abstractions called L-systems, demonstrating that they can produce lifelike plant structures (Lindenmayer 1968).

Initially, both Turing and Lindenmayer drew inspiration from the patterns observed in nature, prior to their endeavors to describe the mechanisms behind these patterns. They took opposite perspectives on development: Turing's cell-chemistry is a bottom-up approach whereas Lindenmayer's grammatical systems are top-down. Interestingly, neither one of those were designed to be evolved, nor were they intended specifically to explain how neural networks are constructed. However, both serve as biological motivation for neuroevolution that takes advantage of indirect encoding through development. This section focuses on approaches based on cell chemistry; the next section focuses on grammatical approaches.

Cell-chemistry approaches aim to capture and utilize some of the fundamental physical mechanisms underlying development. Turing's reaction-diffusion model is a foundation for many of them. It consists of differential equations that describe how chemical substances, or morphogens, propagate and change over time through diffusion through a medium and reaction with each other. Initially the morphogens are randomly distributed, and their concentration vector C at each location changes over time as

$$\partial C/\partial t = F(C) + \mathbf{D}\nabla^2 C, \tag{4.20}$$

where the diagonal matrix $\mathbf{D}$ represents how fast each morphogen diffuses through the medium and the function F describes how the morphogens react to each other. The process characterized by this equation takes place at all locations and time steps in parallel, resulting in a dynamic system of morphogen concentrations. Over time, it can result in significant patterns such as those on seashells, feathers in birds, and fur in mammals.

The model can be applied to the development of neural networks as well (Nolfi and Parisi 1994b). Diffusion represents axonal growth and reactions are interactions between axons and cell bodies, i.e. the forming of active connections. To evolve networks, each genome of a network consists of its neuron definitions, i.e. the location of each cell body and parameters that define how axons will branch out of it. There is exuberant growth with pruning to remove connections that are not useful. In this manner, reaction-diffusion implements a developmental mechanism that allows coding network structures indirectly. It is an abstract analogy, however, i.e. not intended to model the actual underlying chemical processes.

Approaches based on genetic regulatory networks (GRNs), in contrast, aim at building on such chemical processes. As mentioned in the introduction to this chapter, the number of genes in e.g. human genome is relatively small. Much of the complexity lies in the mechanisms that construct an individual based on those genes (GRNs; Wang 2013). In particular, the genes interact: Many genes participate in encoding a particular trait through a complex network of interactions. Through chemical reactions and diffusion, the networks may enhance or suppress the effect of individual genes, generating variation and robustness in gene expression. In this manner, instead of coding everything directly into genes, evolution also encodes an interaction mechanism that results in an indirect and potentially more powerful encoding. Interestingly, this mechanism is entirely missing from standard evolutionary algorithms!

GRNs can be implemented as differential equations, or abstracted into computationally more efficient implementation, such as Boolean functions (Dellaert and Beer 1994). Such functions called operons describe the interactions at a high level, for instance

$$A \wedge \neg B \rightarrow C;$$

$$A \wedge C \rightarrow B,$$

which states that if protein A is in the cell and B is not, then C is produced, and if A and C are both in the cell, B is produced. Thus, starting from A, this process produces C, then B, and stops. Such systems of rules or equations can be encoded as genomes and then evolved towards a given target, such as a production of a certain protein.

Importantly, GRN processes can be scaled up to represent growing neural networks. Some of the protein may represent receptors and others axonal growth. The proteins have

to match in order for the connection to be made. In this manner, chemistry-guided axonal growth like that observed in the brain can be modeled and utilized in neuroevolution. The approach is potential powerful, however it is difficult to take advantage of it. It may need to be simplified further by representing the genome as a string. It can then be evolved to e.g. construct a neural network that controls a simulated robot to move around without hitting obstacles. Or, GRNs may be abstracted into a more general representation of analog genetic encoding, which then allows for complexification and decomplexification of the network as needed in the evolutionary process (Mattiussi and Floreano 2007). Other implementations exist as well (Iba and Noman 2016). A particularly ambitious example will be discussed in Section 9.1.3, where GRNs are used to construct a system with high evolvability, as a potential ingredient in open-ended evolution.

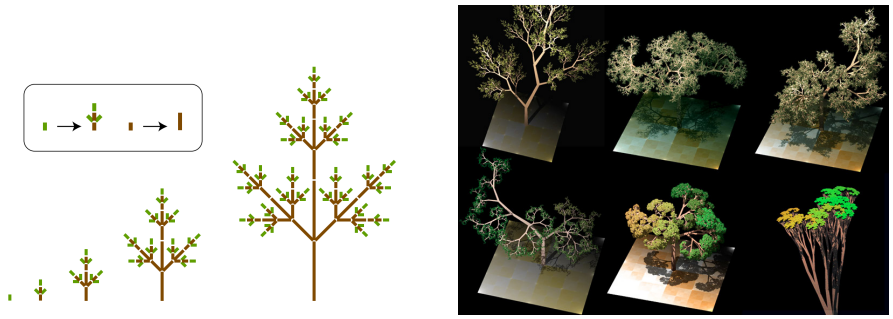
In general, much work remains in taking advantage of indirect encodings through development. A closer look at biological development reveals that between grammatical and cell-chemistry approaches there are many dimensions that could be modeled and utilized (Stanley and Miikkulainen 2003). There are mechanisms for (1) cell fate, i.e. what role each cell develops to take on in the organism; (2) targeting, i.e. how connections find their appropriate end locations; (3) heterochrony, i.e. how timing and ordering of developmental phases affects the end result; (4) canalization, i.e. how some changes because robust and tolerant to mutations; and (5) complexification, i.e. how new genes are added to the genome, increasing the complexity of the phenotype. NEAT, of course, takes advantage of complexification, and GRNs utilize targeting, but the other dimensions and their combinations are largely unexplored.

Thus, much can still be learned from biology and harnessed in neuroevolution. Such work can also help understand biology better, as will be discussed from several perspectives in Chapter 14.

4.2.2 Grammatical Encodings

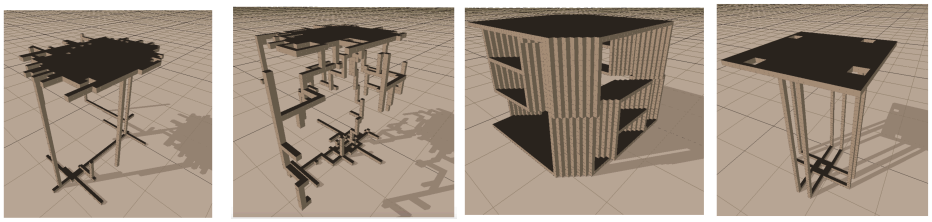
In contrast with the cell-chemistry approaches, Lindenmayer's L-Systems are high-level abstractions of development. They are grammatical rewrite systems; each rewrite step can be seen as a step in development. As mentioned above, they were originally developed to explain patterns seen in plants, and indeed they can produce some very interesting such designs. For instance, the company SpeedTree has created tools that can produce realistic virtual foliage, which has been used in many videos and movies such as Iron Man 3 or Avatar. In L-Systems, rewrite rules are applied concurrently to all characters within a string, similar to how cell divisions occur simultaneously in multicellular organism. By iteratively replacing sections of a basic object according to a predefined set of rewriting rules, intricate structures can be generated. Figure 4.1a shows an example of such a process.

In an impressive demonstration of their versatility, and going beyond the lifelike plant structures they were initially designed for, Hornby and Pollack (2001) applied an L-System approach to the optimization of table designs. While the grammatical rules leading to certain structures are traditionally designed by hand, such as in Lindenmayer's original system, they can also be optimized through an evolutionary search method. For instance, in the case of tables, one can optimize L-System rules that grow designs that have a specific height, surface structure, and stability. Compared to a direct encoding approach, in which discovered components could not be reused, the indirect L-System encoding produced better results



(*a*) L-System Rewriting (*b*)

Figure 4.1: **L-Systems** (a) L-Systems can grow plant-like structures by repeatedly applying rewrite rules to an initial starting character. (b) With the addition of some stochasticity, the approach is able to generate realistic trees. (Figures from Prusinkiewicz et al. 2018) and wikipedia.



(*a*) (*b*) (*c*) (*d*)

Figure 4.2: **Tables grown by evolved L-Systems.** Shown are tables evolved with a direct (a, b) and indirect encoding (c, d). In contrast to the directly encoded tables, the indirectly encoded ones display key biological regularities such as repetition and symmetry. (Figures from Hornby and Pollack 2001).

faster and those designed were more aesthetically pleasing (Figure 4.2) and on quick first glance, could be mistaken for IKEA furniture. On the other hand, the designs produced by the direct encoding approach are lacking regularities and look more piecemeal.

By identifying the shared properties among natural patterns, it becomes evident which aspects artificial systems should account for. One of the fundamental characteristics observed in biological organisms is the presence of repetition. This hallmark trait manifests in multiple instances of the same substructures found throughout an organism's body. From the tiniest cells to complex neural networks in the brain, these recurring motifs play a crucial role in shaping the organism's structure and function. This repetitive nature in the outward appearance of an organism is also referred to as self-similarity. Furthermore, this repetition is not always exact but often occurs with subtle variations. For example, within the vertebral column, each vertebra shares a similarity in structure but exhibits distinct proportions and morphologies. Similarly, human fingers follow a regular pattern, yet they display individual differences, making each finger on the same hand unique. This phenomenon of repetition with variation is pervasive throughout all of natural life. A prevalent form of repetition in biological organisms is through symmetry. Bilateral symmetry, a classic example,

occurs when the left and right sides of an organism's body are mirror images of each other. This symmetrical arrangement is commonly observed in various living beings. While overall symmetry is noticeable in many biological structures, true perfection is rare. Imperfect symmetry is a common feature of repetition with variation. The human body, for instance, exhibits an overall symmetric layout, yet it is not entirely equivalent on both sides. Some organs are exclusive to one side of the body, and the dominance of one hand over the other is a typical example of this asymmetry. In conclusion, the occurrence of repetition and its variations, along with different forms of symmetry, play a fundamental role in shaping the intricate structures and patterns found in biological organisms. Understanding these principles is essential for unraveling the complexities of life and the underlying mechanisms that govern the diversity of living forms.

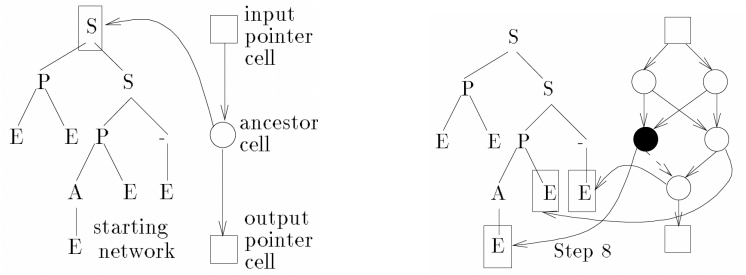
Throughout many generations, the regularities observed in biological organisms often undergo elaboration and further exploitation. An illustrative example of this process is evident in the evolution of early fish, where the bilaterally symmetric fins gradually transformed into the arms and hands of mammals, while still retaining some of the original regularities. Preservation of established regularities is a remarkable aspect of biological evolution. Over generations, these regularities are typically strictly maintained. For instance, bilateral symmetry rarely gives rise to three-way symmetry, and animals with four limbs rarely produce offspring with a different number of limbs, even though the limb design itself may undergo elaboration and modification.

By using this list of regularities and their evolutionary patterns, researchers can analyze phenotypes and lineages resulting from artificial encodings, comparing them to natural characteristics. This analysis provides valuable insights into whether a particular encoding accurately captures the essential properties and capabilities observed in the process of natural development.

The grammatical approach can be applied to neuroevolution as well. In cellular encoding (CE; Gruau and Whitley 1993; Gruau, Whitley, and Pyeatt 1996), a grammar describes how the neural network should be constructed step by step. The process starts with an ancestor cell connected directly to input and output ("cell" here refers to a node in the neural network being constructed; Figure 4.3*a*). Each cell has a pointer to the grammar, which is represented as a tree. Each node in the grammar tree contains an instruction that specifies how the neural network should be modified. After each such step is completed, the pointer traverses to the child of node, until a node with the "end" instruction is reached.

For example in Figure 4.3, the first step is a sequential division. The top cell is then divided in parallel, and the bottom node sequentially again. The top node of that division is divided in parallel, and the connection into the bottom node negated. As the last step, one is added to the threshold of the first node resulting from the last parallel division. As a result of this construction process, a neural network that implements XOR is created (Figure 4.3*b*),

An important extension to this simple example is the ability to include recurrency in the grammar. For example, if a recurrency is added to the leftmost end node, the entire network structure is constructed again at that location from the top of the grammar. Its output becomes the first input of the first network, thus including one more input to the combined network. A counter can then be used to specify that the recurrency should be traversed n times. Thus, the execution of the grammar results in a network that calculates $n+1$-bit parity! Similarly, networks can be constructed that calculate e.g. whether the input



(*a*) Initial network) (*b* Final XOR network)

Figure 4.3: **Cellular encoding approach to evolving neural network structure.** (*a*) The grammar encodes instructions on how to construct the network step by step, starting from a network that consists of a single ancestor cell. Each cell points to a location in the current location in the grammar tree, and is advanced to a child node in the tree as the instruction is executed. S=sequential division, P=parallel division, - = negating a connection, A=adding one to a node threshold, E=end the construction branch. In addition, a recurrency symbol R (not shown) allows continuing the construction again from the top of the grammar, with a counter deciding how many times the recurrency can be traversed. (*b*) After eight steps, the network that results from this construction process implements XOR. With recurrency added to the bottom right of the grammar, it can be extended by repeating the entire structure, thus implementing networks that calculate parity of any number of inputs. The grammar trees can be evolved with genetic programming techniques, making automated discovery of complex networks with repeating structure possible. (Figures from Gruau and Whitley 1993)

vector has a symmetric pattern of ones and zeros. Thus, the recurrency in the grammar is a powerful way to take advantage of repetitive structure in networks.

Whereas L-systems were not designed to be evolved, CE was: Because the CE grammars are trees, genetic programming (Banzhaf et al. 1998) is a natural way to evolve them. Indeed, parity networks upto 51 bits were evolved in this manner, demonstrating that evolution can indeed take advantage of repetition. It is also possible to prove that any neural network topology can be represented in CE grammars. However, it does not mean that the good topologies are easy to find. As a matter of fact, the grammar can be turned around to represent to code connections in the network rather than cells, resulting in a different bias in the kinds of networks that can be constructed easily (Luke and Spector 1996). The challenge is to discover the right biases and code them into the grammatical representation.

Besides L-systems and CE, other grammatical encoding mechanisms have been developed as well. For instance in order to scale neuroevolution to the size and complexity of deep learning, it is possible to represent the weights as a sequence of mutations, and only store the mutation seeds (Such et al. 2017). This approach will be described in more detail in Section 12.4 in the context of scaling up neuroevolution in reinforcement learning tasks.

Thus, encoding the developmental processes as a series of grammatical rewrite operations is a high-level alternative to systems that aim at replicating the low-level cell-chemistry mechanisms. Incorporating learning as a lifetime stage of development synergetically with evolution is a third approach, as will be described next.

4.2.3 Learning approaches

In addition to the physical development explored in the last two subsections, much of biological development happens through learning. The individual interacts with the environment and adapts its structure and parameters accordingly. Such learning is a form of indirect encoding as well: Evolution defines a starting point and a learning mechanism, and the full individual emerges indirectly through their synergy. The biological implications of this idea are explored in more depth in Section 14.4. In this subsection, the synergy is put to work as a computational mechanism that allows us to construct more complex systems.

Many of the neuroevolution methods reviewed so far can be used to construct the initial starting point, and many of the standard neural network learning algorithms used to establish the developmental phase. But several questions remain: First, should the improvements discovered by learning be coded back into the genome, in a Lamarckian evolutionary process, or should it only determine the fitness of the individual, thus guiding a Darwinian evolution through the Baldwin effect? Second, if gradient-descent-based learning methods are to be used, where do the targets for it come from? Third, does the development require weight adaptation, or can it be more effectively encoded as a state of activation? Each of these questions is addressed in turn in this section.

First, Lamarckian evolution (Lamarck 1809/1984) suggests that acquired traits can be inherited, which is unlikely in biology. For instance, giraffes stretch their necks in order to reach higher, and their offspring will have longer necks as a result. In some cases, non-genetic transmission is possible through epigenetic means (Lacal and Ventura 2018). For instance in a process called methylation, a methyl molecule attaches to the DNA, modulating genetic expression. As a result, for instance animals that must live in a hostile environment may have offspring that are more sensitive and fearful, compared to offspring of those who exist in a normal environment. While such changes are not permanently encoded in the DNA, they do provide an immediate survival advantage that is inheritable.

Whether biologically plausible or not, computational evolution can take advantage of both Lamarckian evolution and epigenetics. For instance, it may be possible to take advantage of these principles in evolving deep learning networks. Such networks are often too large to evolve effectively; however, it may be possible to train them and code the learned weights back to the genome. This approach has been successful for instance in evolving convolutional architectures for image processing (Prellberg and Kramer 2018; Hadjiivanov and Blair 2021). Through the approach, evolutionary exploration and gradient-based tuning can be combined.

One challenge in implementing Lamarckian/epigenetic evolution is that it may lead to a loss of diversity. Through gradient descent, the individuals in the population are modified in the same direction, as suggested by the gradient. The learning process may thus interfere with evolutionary exploration. A possible way to cope with this challenge is to train different individuals with different batches of data, or more broadly, use ensembling techniques to keep the population diverse. Effective ways of managing exploration and learning are still open to research.

The Baldwin effect can also lead to powerful computational approaches. The adaptations are not coded back into the genome, but only used to determine fitness. Learning thus guides evolution towards more promising individuals (which is the Baldwin effect). Indeed early studies showed that such a combination can be more powerful than evolution or learning

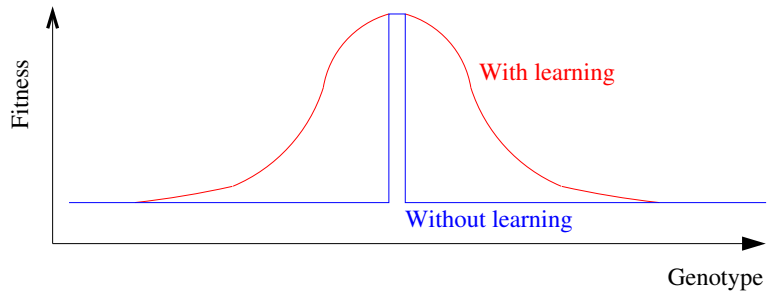


Figure 4.4: **Learning guiding evolution through the Baldwin effect.** In this needle-in-the-haystack problem, it would be difficult for evolution to find the sharp peak when the fitness evaluations of the other solutions are all the same. However, learning allows modifying these solutions, i.e. moving left and right along the x-axis. Therefore, the closer the solution is to the peak, the more likely it is to find it through learning, as indicated by the red curve. Learning can thus provide a more useful fitness, and help evolution find the peak faster. (Adapted from Hinton and Nowlan 1987)

alone. For instance, in the needle-in-the-haystack problem, even when learning consisted of simply random changes, it was enough to broaden the basin of the target, and make it more likely for evolution to discover it (Figure 4.4; Hinton and Nowlan 1987). Thus, even if the learning does not affect the genome, it can be useful in guiding the evolution by suggesting which genetic individuals are more promising.

Interestingly, this result does not mean that an evolutionary system guided by the Baldwin effect gradually encodes more and more of the learned information into the genes, eventually making learning unnecessary. That is, the evolved solutions before learning often perform quite poorly—it is only after the learning that they perform well. This phenomenon is precisely the idea of synergetic development. Because learning is always part of the evaluation, evolution discovers the best possible starting points for learning, so that the system as a whole performs as well as possible. The starting points can be far from the optimum as long as learning can reliably pull them into the optimum. Apparently, in many tasks there are many such starting points and they are easier for evolution to find than points close to the optimum would be. Therefore, evolution finds a synergy where both methods play a significant role.

Regarding the second question posed in the beginning of this subsection, so far the discussion has assumed that the optimal targets for gradient descent are known. However, surprisingly the process works even when such targets are not available. One possibility is to use related targets, such as predicting how the inputs are going to change as a result of the action as targets for backprop (Section 14.4.1). They do not directly specify what the agent should do, but they do allow learning internal representations that help evaluate the candidate.

Another approach is to use the behavior of current population champions, or even just that of parents, to train the offspring (McQuesten and Miikkulainen 1997). This result is counterintuitive, because evolution depends on discovering offspring that is better than the parents. However, what is important is that the offspring performs well after training. Thus,

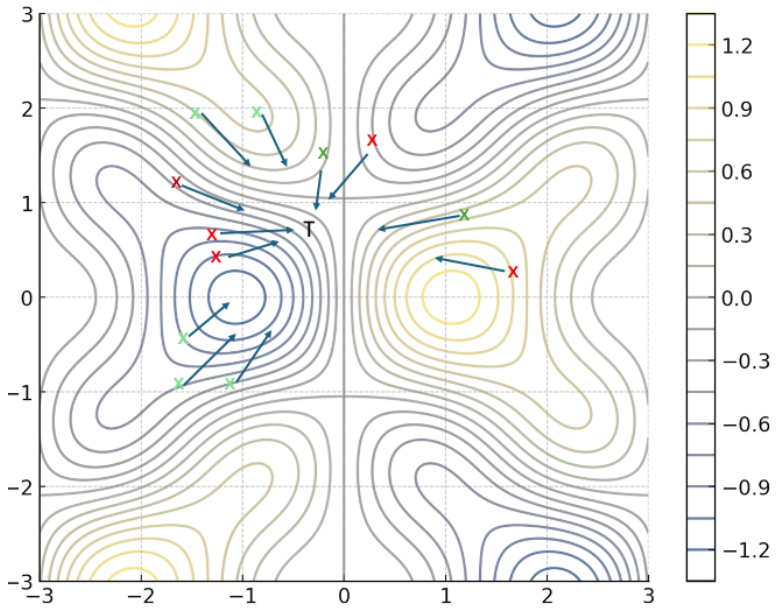


Figure 4.5: **Training to imitate champions or parents.** When well-performing individuals, such as population champions or parents, are used as teachers (T), they pull the offspring (X) towards the teachers. Those offspring that perform the best after training are likely to be located near the optimum to begin with, and although some (red X) are worse after training, some (green X) are likely pulled closer to the optimum. Such training provides useful exploration around the optimum, making it more likely to be discovered.

the process takes advantage of the Baldwin effect in the same way as evolution did in the needle-in-the-haystack problem (Figure 4.4; Hinton and Nowlan 1987). If the teachers are in the neighborhood of the optimal solutions, training will move the offspring around in this neighborhood, making it more likely that some of them will get closer to the optimum (Figure 4.5). Selecting such solutions allows evolution to make progress even when the fitness evaluations without learning are not very informative.

The third question concerns the nature of adaptation: Is it necessary to encode the learned behaviors into the weights, or could it be more effective to encode them simply as a recurrent activation state? Of course, if the network needs to perform many trials starting from a reset activation, weight adaptation is necessary. However in many domains, individuals perform and adapt continuously throughout their lifetime. With the appropriate recurrent circuitry, they could develop an activation state that modulates their further actions, similarly to a change in weights. Such an encoding of adaptation could be easier to discover and maintain.

To study this question, instead of gradient descent, a more general and low-level adaptation mechanism is needed: Hebbian learning (Widrow et al. 2023). The basic idea is that if the neurons on both sides of the connection are active at the same time, the connection is useful and its weight should be increased. To bound such increases, a normalization process

such as weight decay is also needed, for instance:

$$\Delta w_{ij} = \alpha_{ij} o_i o_j - \beta_{ij} w_{ij}, \tag{4.21}$$

where w_{ij} is the weight between neurons i and j with activations o_i and o_j, and α_{ij} and β_{ij} are learning and decay rate parameters. Unlike gradient descent, Hebbian learning is entirely local to each connection and requires no learning targets at the output. In this sense, it is closer to biological learning than gradient descent, and therefore a proper comparison to adaptation based on recurrency. Note that Hebbian learning also provides an alternative that avoids the second question in this section, i.e. where the targets for development come from—it does not need them. On the other hand, it cannot take advantage of targets either, and therefore it is generally not as powerful as gradient descent.

Nevertheless, Hebbian learning is a compelling approach to developmental indirect encoding on its own. Networks with Hebbian learning can change their behavior based on what they observe during their lifetime. For instance, they can evolve to first perform one task such as turn on a light, and then switch to another such as travel to a target area (D. Floreano and J. Urzelai 2000). While it is biologically plausible, an interesting practical question arises: Can such low-level adaptation be more effectively implemented through recurrent activation?

The above foraging domain with good and bad food items can be used to study this question (Stanley, Bryant, and Miikkulainen 2003). The usual NEAT method for evolving recurrent networks can be compared with a version that takes advantage of Hebbian learning: It evolves the learning rate and decay rate parameters α_{ij} and β_{ij} for each connection, in addition to the weights and the network topology. Each evolved network is placed into the foraging environment where it can consume food items; if an item is good, it receives a pleasure signal, and if bad, a pain signal. All items in a trial are the same so after it consumes the first item, it needs to either eat all of them or none of them to receive maximum fitness.

While both approaches evolved successful networks, NEAT without adaptation required about half the generations to do so. There were fewer parameters to optimize, and evaluations were more consistent. Indeed the solution networks look very different (Figure 4.6): While the fixed-weight recurrent networks were parsimonious with recurrency focused at the output, the adaptive networks were more complex and holistic, using many more adaptive weights throughout the network. Because many weights adapt, it was not possible to rely on only a few loops, and the behavior became encoded redundantly throughout.

Thus, in such a simple task recurrency was more effective than Hebbian adaptation. It is of course possible that in more complex situations adaptation provides additional power that may be needed. What such tasks might be is an interesting topic for future work.

4.3 Indirect Encoding Through Hypernetworks

A common features of the indirect encodings we have encountered in the previous sections is that a specific phenotypic component at a given point in development influences the states of nearby components. In other words, here development progresses through local interactions. This section reviews a particularly popular indirect encoding that, when