Evolving Weights in a Fixed Network

David Montana and Lawrence Davis (1989) took the first approach—evolving the weights in a fixed network. That is, Montana and Davis were using the GA *instead* of back-propagation as a way of finding a good set of weights for a fixed set of connections. Several problems associated with the back-propagation algorithm (e.g., the tendency to get stuck at local optima in weight space, or the unavailability of a "teacher" to supervise learning in some tasks) often make it desirable to find alternative weight-training schemes.

Montana and Davis were interested in using neural networks to classify underwater sonic "lofargrams" (similar to spectrograms) into two classes: "interesting" and "not interesting." The overall goal was to "detect and reason about interesting signals in the midst of the wide variety of acoustic noise and interference which exist in the ocean." The networks were to be trained from a database containing lofargrams and classifications made by experts as to whether or not a given lofargram is "interesting." Each network had four input units, representing four parameters used by an expert system that performed the same classification. Each network had one output unit and two layers of hidden units (the first with seven units and the second with ten units). The networks were fully connected feed-forward networks—that is, each unit was connected to every unit in the next higher layer. In total there were 108 weighted connections between units. In addition, there were 18 weighted connections between the non-input units and a "threshold unit" whose outgoing links implemented the thresholding for each of the non-input units, for a total of 126 weights to evolve.

The GA was used as follows. Each chromosome was a list (or "vector") of 126 weights. Figure 2.17 shows (for a much smaller network) how the encoding was done: the weights were read off the network in a fixed order (from left to right and from top to bottom) and placed in a list. Notice that each "gene" in the chromosome is a real number rather than a bit. To calculate the fitness of a given chromosome, the weights in the chromosome were assigned to the links in the corresponding network, the network was run on the training set (here 236 examples from the database of lofargrams), and the sum of the squares of the errors (collected over all the training cycles) was returned. Here, an "error" was the difference between the desired output activation value and the actual output activation value. Low error meant high fitness.

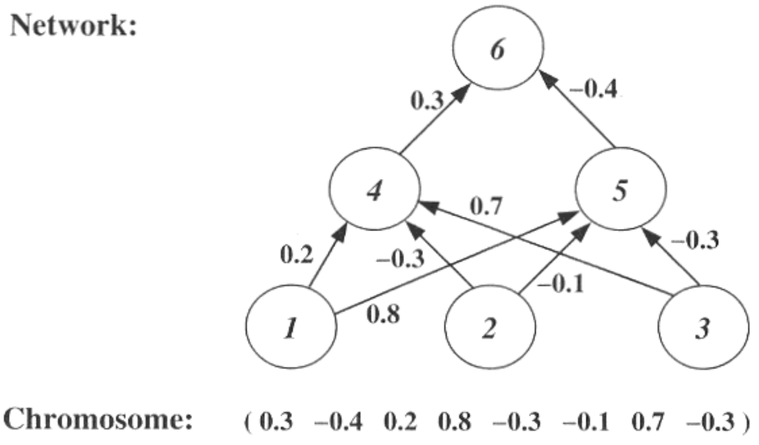

Figure 2.17 Illustration of Montana and Davis's encoding of network weights into a list that serves as a chromosome for the GA. The units in the network are numbered for later reference. The real-valued numbers on the links are the weights.

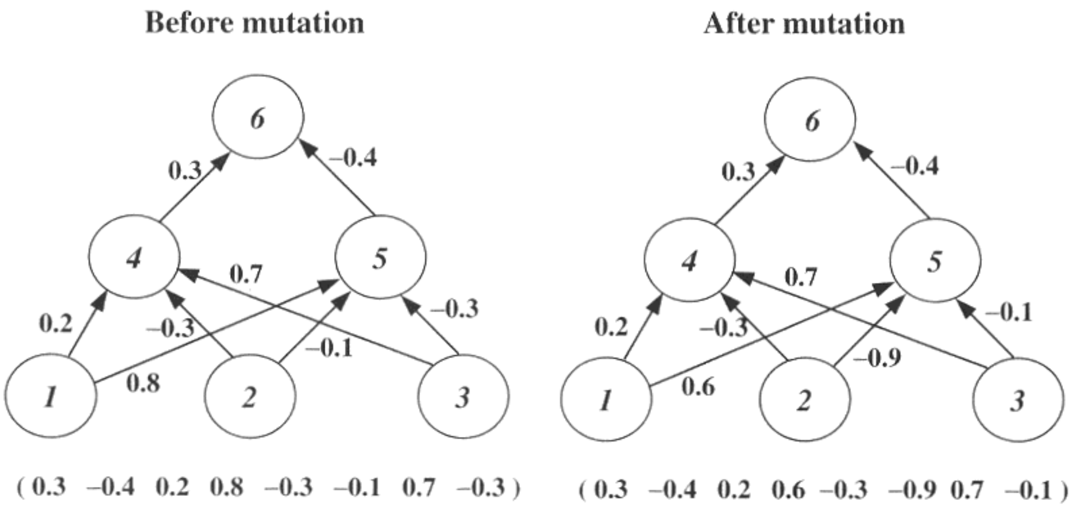

Figure 2.18 Illustration of Montana and Davis's mutation method. Here the weights on incoming links to unit 5 are mutated.

An initial population of 50 weight vectors was chosen randomly, with each weight being between −1.0 and +1.0. Montana and Davis tried a number of different genetic operators in various experiments. The mutation and crossover operators they used for their comparison of the GA with back-propagation are illustrated in figures 2.18 and 2.19. The mutation operator selects *n* non-input units and, for each incoming link to those units, adds a random value between −1.0 and +1.0 to the weight on the link. The crossover operator takes two parent weight vectors and, for each non-input unit in the offspring vector, selects one of the parents at random and copies the weights on the incoming links from that parent to the offspring. Notice that only one offspring is created.

The performance of a GA using these operators was compared with the performance of a back-propagation algorithm. The GA had a population of 50 weight vectors, and a rank-selection method was used. The GA was allowed to run for 200 generations (i.e., 10,000 network evaluations). The back-propagation algorithm was allowed to run for 5000 iterations, where one iteration is a complete epoch (a complete pass through the training data). Montana and Davis reasoned that two network evalua-

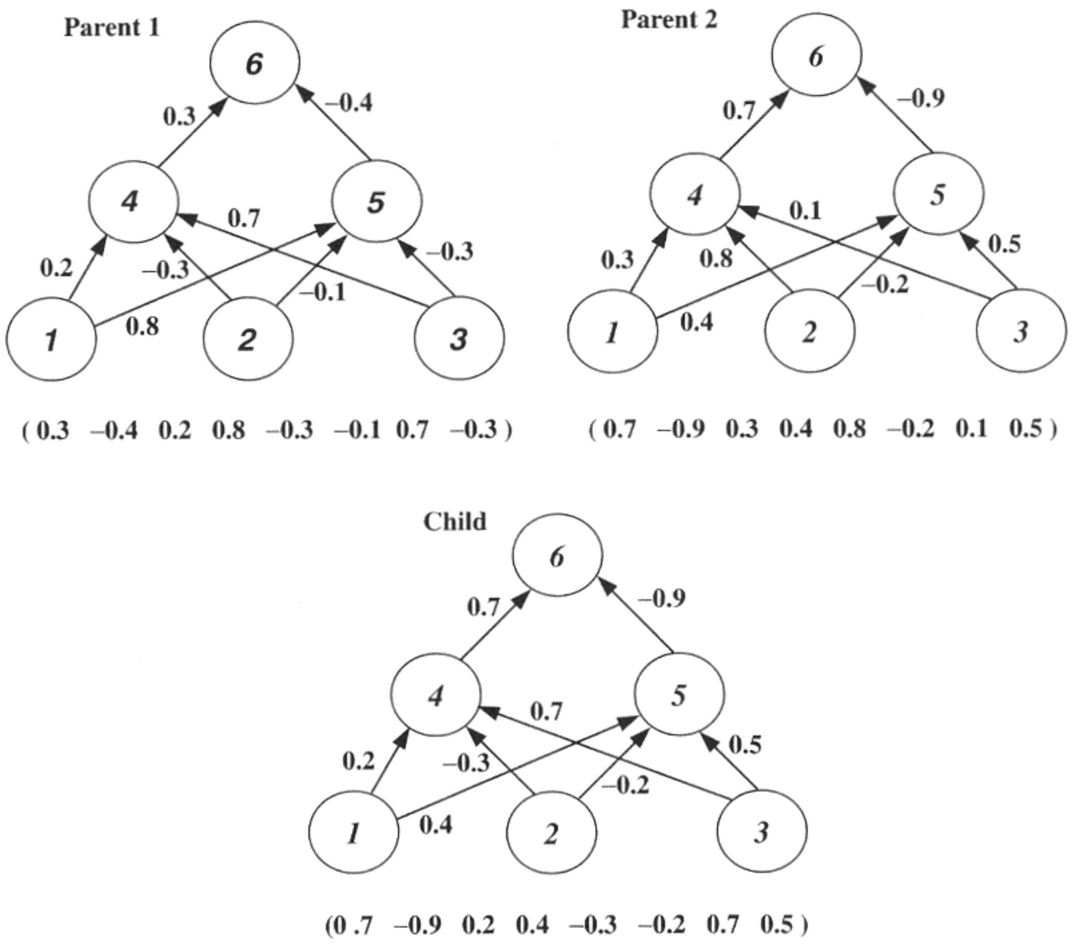

Figure 2.19 Illustration of Montana and Davis's crossover method. The offspring is created as follows: for each non-input unit, a parent is chosen at random and the weights on the incoming links to that unit are copied from the chosen parent. In the child network shown here, the incoming links to unit 4 come from parent 1 and the incoming links to units 5 and 6 come from parent 2.

tions under the GA are equivalent to one back-propagation iteration, since back-propagation on a given training example consists of two parts—the forward propagation of activation (and the calculation of errors at the output units) and the backward error propagation (and adjusting of the weights). The GA performs only the first part. Since the second part requires more computation, two GA evaluations takes less than half the computation of a single back-propagation iteration.

The results of the comparison are displayed in figure 2.20. Here one back-propagation iteration is plotted for every two GA evaluations. The x axis gives the number of iterations, and the y axis gives the best evaluation (lowest sum of squares of errors) found by that time. It can be seen that the GA significantly outperforms back-propagation on this task, obtaining better weight vectors more quickly.

This experiment shows that in some situations the GA is a better training method for networks than simple back-propagation. This does not mean that the GA will outperform back-propagation in all cases. It is also possible that enhancements of back-propagation might help it overcome some of the problems that prevented it from performing as well as the GA in this experiment. Schaffer, Whitley, and Eshelman (1992) point out

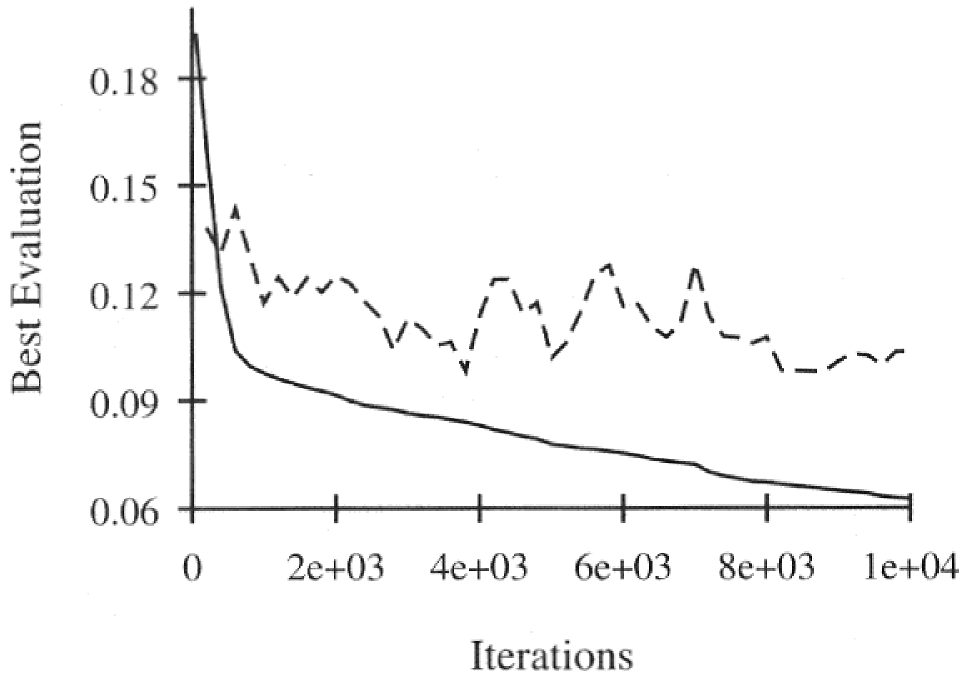

Figure 2.20 Montana and Davis's results comparing the performance of the GA with back-propagation. The figure plots the best evaluation (lower is better) found by a given iteration. Solid line: genetic algorithm. Broken line: back-propagation. (Reprinted from *Proceedings of the International Joint Conference on Artificial Intelligence;* © 1989 Morgan Kaufmann Publishers, Inc. Reprinted by permission of the publisher.)

that the GA has not been found to outperform the best weight-adjustment methods (e.g., "quickprop") on supervised learning tasks, but they predict that the GA will be most useful in finding weights in tasks where back-propagation and its relatives cannot be used, such as in unsupervised learning tasks, in which the error at each output unit is not available to the learning system, or in situations in which only sparse reinforcement is available. This is often the case for "neurocontrol" tasks, in which neural networks are used to control complicated systems such as robots navigating in unfamiliar environments.