Does Data-Model Co-evolution Improve Generalization Performance of Evolving Learners?

J. L. Shapiro

University of Manchester, Manchester M13 9PL UK.

Abstract. In co-evolution as defined by Hillis, data which defines a problem co-evolves simultaneously with a population of models searching for solutions. Herein, one way in which this could work is explored theoretically. Co-evolution could lead to improvement by evolving the data to a set which if a model is trained on it, it will generalize better than a model trained on the initial data. It is argued here that the data will not necessarily co-evolve to such a set in general. It is then shown in an extremely simple toy example that if there is too much optimization per generation, the system oscillates between very low fitness solutions, and will perform much worse than a system with no co-evolution. If the learning parameters are scaled appropriately, the data set does evolve to one which leads to better generalization performance. The improvement can be arbitrarily better in the large population size limit, but strong finite-population effects limit what can be achieved.

1 Introduction

Several years ago, Hillis [1] introduced a co-evolution scheme in which, while potential solutions to a problem adapt, the sample data which defines the problem co-adapts. Hillis was evolving sorting networks for sorting lists of 16 elements. The goal was to find a sorting network which sorts all lists of 16 elements with the least number of comparisons. The fitness of a network was the number of lists correctly sorted from a sample set of lists. The lists in the sample set co-evolved, where the fitness of a list was the number of sorting networks in the population which failed to sort it correctly. Using this procedure, Hillis found a sorting network which was better than that found using fixed data, and it was close to the best that was known for lists of length 16. Although there were other differences between the co-evolving and non-co-evolving experiments, Hillis argued that it was the co-evolution which led to the increase in performance.

Hillis' scheme, which I will call *data-model co-evolution* to distinguish it from other so-called co-evolutionary algorithms, has generated a great deal of recent interest, for obvious reasons. If it works in general, it could be applied to any problem which requires performance on a large number of cases, but can only be tested on a small fraction of those cases. This is a huge class of enormous importance. In addition, the approach could be applied in a wide range of algorithms.

In fact, any algorithm which can produce a population could be used to evolve the models or the data. So, in addition to evolutionary algorithms, it could be used in parallel random hill-climbing or simulated annealing in which a population is generated at each timestep via different realizations of the stochasticity. In fact, it gives a new reason to have a population of models in an algorithm; a population of models is used as a sample against which the performance of the data can be measured and improved.

There have been a number of experimental studies of this type of system. For an overview, see [2]. In this paper, a particular theoretical question will be considered. Does data-model co-evolution work by evolving the dataset to one which leads to better true or generalization performance? In the types of problems which uses a data set to define the problem, it is usually the case that the dataset contains only a tiny fraction of possible cases. The performance of a model on this data set is an estimate of the true performance, which means the performance on all possible data. Good performance on some datasets may lead to better true performance than good performance on other datasets. Is it possible that data-model co-evolution improves the performance of the models by evolving the data to such a set? A similar question was considered in [5] in the context of game playing.

2 Description of the algorithm

Data-model co-evolution is defined as follows. Let x denote a training example (data) and s denote a possible solution to the problem (model); X denotes a population of N_d data points, and S likewise to refer to a population of N_s solutions.

The fitness of a model is the sum of the fitness for each datum, $f(s, x)$. The apparent fitness or training fitness is usually taken as,

$$F_a(s) = \sum_{x \in X} f(s, x). \tag{1}$$

The true fitness or generalization fitness is given by

$$F_t(s) = \int f(s, x) P(x) dx, \tag{2}$$

where $P(x)$ is the distribution of data for the problem. The apparent fitness is a noisy and biased estimate of the true fitness, but it is the true fitness one wishes to optimize. Of course, only the apparent fitness is known.

The fitness of a datum, according to Hillis, is the number of models which gets it wrong, i.e.

$$coF_a(x) = \sum_{s \in S} [1 - f(s, x)] \tag{3}$$

The co-evolution algorithm is pair of evolutionary algorithm which alternates between the two populations. The data-set and model-set evolving algorithms can be any algorithms which produce a population of solutions: a genetic algorithm, a parallel simulated annealing algorithm, a random hill-climbing algorithm, etc.

3 Is data-model co-evolution a generalized query construction algorithm?

In order to find mathematical models to study how co-evolution might work, it is useful to decide how it *could* work. I suggest that there are three possibilities. First, it could work by increasing diversity in the population thereby preventing premature convergence. Second, it could work by dynamically changing the shape of the fitness function to make the problem easier. Finally, it could work by evolving the dataset to one which, if learned, gives better true (generalization) performance. Of course, it could be a combination of these.

The last hypothesis is considered here. This hypothesis puts this algorithm in the context of non-evolutionary learning algorithms for adapting a training set called *learning from examples and queries* in learning theory [6]. In this scenario, there is a "teacher" (sometimes called an oracle) which presents labelled examples of members of different classes; the learner can also present examples to the teacher to be classified. The examples which are presented to the teacher are those whose inclusion in the training set most (in some sense) improves the expected true performance. Such an algorithm can improve the performance over random examples considerably.

Query learning algorithms are always derived for a specific learning system and specific class of problem. Data-model co-evolution is completely general, however. No information about the problem or class of potential solutions is used. Instead, search is used to find the best data. This generality means it could be quite powerful if we could show that it works as a query construction algorithm. (Another difference with usual query learning is data is only added and never removed.)

We can ask the question this way, does adding a point to X which optimizes the co-fitness of equation 3 lead to an increase in the expected true fitness

$$F_t(X+x) = \int P(s|X+x)F_t(s)ds \tag{4}$$

where the integral is over the distribution from which S is drawn (assuming the population can be expressed in this way).

In principle the answer is no. First, if there is no model which exactly solves the problem, due to noise for example or to an insufficiently complex model class, this will not lead to an increase in true performance. For this situation, the cases must remain representative of their distribution to find an optimal solution. For example, in regression, the data-set will evolve to one filled with the most extreme values, which may not give an optimal result. This point may well hold for ordinary query learning.

More generally, however, this will not lead to improvement necessarily even if perfect performance is possible. Consider the following situation. Suppose in the population there are models all with optimal *apparent* fitness. One of the models has perfect true performance. Many of the models get most of the possible data correct, but get a few wrong that are not in the current data population, and it is the same few. The remaining model gets most of the data outside current data set wrong, but not those few. Which datum should be added next. Hillis's heuristic will add a datum which eliminates the many good solutions, since it is trying to beat as many models as possible. However, to improve the expected true performance it is best is remove the one model whose true performance is bad by adding a datum which removes it (remember that the apparent fitness of all is the same, so the solution which is finally used is drawn uniformly from this population). But this will only beat one model, and so has a low fitness according to Hillis.

So this approach does not lead to improvement of generalization in all cases. It is more like an entropy heuristic; it decreases the volume of acceptable models if not their expected performance.

4 A simple example – one dimensional high-low

Although it cannot be proved to work in principle, it could still work in practice. A very simple example is now considered to see whether the data set will co-evolve to one which leads to increased true performance.

4.1 Description of the example problem

We consider Gibbs learning in the simplest toy problem from computational learning theory — the one-dimensional "high-low" problem. An instance of the problem is defined by a number $t \in (0, 1)$. In this model, a number $x \in (0, 1)$ is classified as 1 if $x > t$ and 0 otherwise. The learning problem is to make a model of the data from a set of examples $\{x_i\}$ and the classes they are in. Usually one assumes that the learning system comes from the same class as the problem. Thus, the learning system is trying to find t from a set of examples.

The optimal solution space lies between the examples which bound t ¿from either side. Let

$$x_R = \min \{x_i | x_i > t\} \tag{5}$$

$$x_L = \max \{x_i | x_i \leq t\}. \tag{6}$$

If learning is such that the optimal subspace is reached, true performance will be related to

$$e_g = x_R - x_L. \tag{7}$$

It is easy to show that on average, $e_g \approx 2/N_d$, where N_d is the number of training examples. Thus, this system has the same relationship between training set size and generalization as the perceptron and other simple systems. In addition, it is easy to generate a query learning algorithm for this system. To add a new example, simple pick an x between x_R and x_L. This increases the true performance exponentially, as it is essentially binary search.

Thus, this seems a good example to test the hypothesis that co-evolution can lead to an example set which produces higher generalization, because there is such a set for the system to evolve to. This is different from the perceptron, for example, in which query learning has a much weaker effect.

4.2 The co-evolutionary learning algorithm

The dynamics of co-evolving systems is very difficult to analyse. Although the system is Markovian, the correlation between the fitness function (through the data) and the current population (through its correlation with previous generations) makes the analysis very difficult [4]. I will use a theoretical device to remove this correlation. Rather than learning for a particular number of steps, the system will learn to a particular distribution defined by a learning parameter, namely a Gibbs distribution with inverse temperature β. This is not realistic and removes the opportunity to study dynamics, but makes the problem solvable. Since we are interested in a property of the dataset which can be considered without dynamics, it is a reasonable first step. This type of learning is called Gibbs learning, and is roughly equivalent to learning via simulated annealing by annealing to the inverse temperature β. The population is generated by sampling from this distribution at each generation.

4.3 Zero temperature Gibbs co-evolutionary learning

First we will consider the following learning scenario. There is a population of models of size N_s and a population of data of size N_d. At each stage, evolution will occur sufficiently long that whichever system is evolving will evolve to the optimal subspace with regard to the other system. Thus, there will be a large number of evolution steps for each system before the other system starts to evolve. The population after the evolution will be drawn independently ¿from this optimal subspace.

For learning systems, this is called “zero temperature Gibbs learning”, because the distribution from which the learners are drawn is the Gibbs distribution in the limit that the temperature goes to zero.

For the models, the optimal subspace lies between x_L and x_R as mentioned in the previous section. Thus, the population of models after evolution will be drawn uniformly between this interval. If it happens that all of the data lies on one side of t, then the missing x_L or x_R is replaced by 0 or 1, but this is very unlikely unless N_d is very small.

For the data, however, the optimal subspace is one of these intervals

$$(t, s_R) \quad \text{if more of the } s\text{'s are greater than } t; \tag{8}$$

$$(s_L, t) \quad \text{if more of the } s\text{'s are less than } t; \tag{9}$$

$$(s_L, s_R) \quad \text{there are the same number of } s\text{'s on either side of } t; \tag{10}$$

Here s_L, s_R are, as with the x's, the closest models of t on either side. Thus, the fitness for the data is maximal on side of t where most of the data is, unless there are exactly the same number of models on each side of t, which will rarely happen.

Thus, the result of the evolution is that the models move to populate the optimal subspace of the initial population. The data then moves to the side of t where more of the models are, and closer to t than they are. The models then move to the side of t where the data is not. The data then moves to where the models are, the models to where the data is not, etc. And so the system will oscillate forever. The true performance measure will oscillate between t and $(1-t)$ plus terms of order $1/N_d$. On average, the true performance after the first step will be $1/2$.

To summarize,

$$\langle F_t \rangle_{pop} = \begin{cases} \frac{2}{N_d}; & \text{without co-evolution} \\ \frac{1}{2}; & \text{with co-evolution} \end{cases} \tag{11}$$

The true fitness with co-evolution is much worse than what you get without co-evolution. It might be tempting to describe this as an example of the *Red Queen Effect* [3], where the system expends its evolutionary resources in maintaining its current fitness rather than in improving it. However, here the system actually gets much worse. This system oscillates between very low fitness states, and the reason for this is transparent. The data occupies an subspace which is optimal against the current population, but leaves other regions of space open for the models to occupy.

4.4 Finite temperature co-evolutionary learning

Obviously, it is necessary to populate sub-optimal as well as optimal subspaces. So, consider what happens if the data learns to a finite temperature Gibbs distribution with inverse temperature β_d. This parameter controls the amount of co-evolution. If β_d is 0, the Gibbs distribution is just a random distribution. The next generation of models will learn against a new batch of data, but one which will be statistically equivalent to the previous data set. As β_d is increased, it will correspond to increasing the amount of co-evolution. Of course as β_d gets increasingly large, the system will approach the oscillating system described in the previous section.

The models will continue to learn to the optimal subspace, as that did not contribute to the oscillations above, and it does make the analysis easier. Thus, the models will be uniformly distributed between x_L^0 and x_R^0, where these are the x's closest to t before the co-evolution step under consideration. Since the true fitness of the models is given by $x_R^0 - x_L^0$ before co-evolution and $x_R - x_L$ after co-evolution, we will consider how $x_R - x_L$ depends upon $x_R^0 - x_L^0$, N_d and N_s. Since learning is equivalent at each generation, this function will allow us to determine the entire sequence of intervals.

The question of interest is, will the oscillating state be preserved for less extreme amounts of co-evolution, or will the system stabilize? The answer is that it depends upon how the fitness is scaled with regard to the learning parameter β_d. An analogy that one can make is that the data population is like an ideal gas in a one-dimensional potential. Such a system cannot have a phase transition except at zero and infinite temperatures. Thus, the finite temperature behaviour must be the same as the zero temperature behaviour or as the infinite temperature behaviour. We know the zero temperature behaviour is oscillatory, so we desire that the phase transition occur at zero temperature.

We assume that the two populations are large. If we assume that the fluctuations in x are larger then the spacing between the levels,

$$\left\langle (x - \langle x \rangle_{\beta_d})^2 \right\rangle_{\beta_d} > \frac{1}{N_d} \tag{12}$$

where $< \cdot >_{\beta_d}$ denotes averaging over the Gibbs distribution, then the discreteness of the levels can be ignored. In this case, the fitness of the data can be approximated as,

$$coF_a(x) = \begin{cases} 0; & x < x_L^0 \\ N_s(x - x_L^0); & x \in (x_L^0, t) \\ N_s(x_R^0 - x); & x \in (t, x_R^0) \\ 0; & x > x_R^0. \end{cases} \tag{13}$$

We see that the gap between the two sides of t is of order N_s, because we count the number of models that the data defeats. With this scaling of the fitness, the system will always be oscillatory for large population size. (In this type of learning, β controls the trade-off between optimization and entropy, with β multiplying the cost function. The loss in fitness by populating the subspace on the other side of t is $O(N_s)$, whereas the entropy is only $\log(N_d)$. Thus the suboptimal subspace on the other side of t is only populated if β is very small, of order $N_s / \log(N_d)$.)

However, if co-fitness is divided by N_s, this system will be in the low temperature phase for finite β. This is the state we will investigate, as it is the more promising scaling. The transition to the oscillating state will only occur at $\beta_d = \infty$ for large population size.

We draw a set of x's from a Boltzmann distribution,

$$Z^{-1} \exp\left[\beta_d coF_a(x)\right] \tag{14}$$

where Z is the normalization. Then we generate the distribution of the closest value below t and above t from a set of N_d drawn from this distribution in order to compute the

$$P(x_L, x_R | x_L^0, x_R^0, N_d) \tag{15}$$

the probability distribution for the new closest values to t in terms of the old ones. Using this to average of the true fitness measure and taking the large N_d limit produces this expression,

$$\begin{aligned} &\langle x_R - x_L \rangle = \\ &\frac{1}{N_d \beta} \left\{ \exp\left[-\frac{e^{\beta\delta_{Rt}}}{\beta}\right] E_i\left[-\frac{e^{\beta\delta_{Rt}}}{\beta}\right] + \exp\left[-\frac{e^{\beta\delta_{tL}}}{\beta}\right] E_i\left[-\frac{e^{\beta\delta_{tL}}}{\beta}\right] - \right. \\ &\left. E_i\left(\frac{1}{\beta}\right)\left[\exp\left(-\frac{e^{\beta\delta_{Rt}}}{\beta}\right) + \exp\left(-\frac{e^{\beta\delta_{tL}}}{\beta}\right)\right]\right\} + \\ &\frac{1}{N_d}\left\{\exp\left[-\frac{e^{\beta\delta_{Rt}-1}}{\beta}\right] + \exp\left[-\frac{e^{\beta\delta_{Rt}-1}}{\beta}\right]\right\} \end{aligned} \tag{16}$$

Here

$$\delta_{Rt} \equiv N_d(x_R^0 - t) \tag{17}$$

$$\delta_{tL} \equiv N_d(t - x_L^0) \tag{18}$$

$$E_i(\lambda) = \int_{-\infty}^{\lambda} \frac{e^t}{t} dt \tag{19}$$

The δ's are order 1, and the principal part of the integral is taken.

For β_d small, this goes to $2/N_d$ as expected. For large β_d, assuming the large N_d limit has been taken first, this goes to zero like $\exp(-\beta_d \delta_{max})/N_d$, where δ_{max} is the maximum of the two intervals. Thus, it is the case that the data evolves to a training set which leads to greater true performance, and with appropriately chosen parameters one can get to arbitrarily high increase in performance through co-evolution. However, it is sensitive to the scaling of the learning parameter.

Finite population effects are very important. There are two types. Finite N_d will lead to a crossover to the oscillatory behavior. This will come in when

$$\beta_d \approx \log N_d. \tag{20}$$

Thus, this will be important unless the population size is very large. Finite N_s will mean that the generalization performance due to co-evolution cannot continue to increase. There is no reason for x_L to get closer to t then the s which is closest to t. This will become important when

$$\beta_d \approx \log N_s. \tag{21}$$

Figure 1 shows a plot of theoretical results against experiments. The solid curves are theoretical curves for the true performance before co-evolution (which is independent of β) and after, according to equation 16. The experiments were done by directly sampling from the Gibbs distribution. This will not tell anything about the time to reach the distribution, since direct sampling can only be done if t is known. The diamonds show the true performance of the models before co-evolution; the asterisks are after. Also shown are the true performance with no co-evolution but with a training set twice as large (dashed curve), three times as large (dot-dashed curve), and four times as large (dotted curve). With co-evolution more data must be processed. In order for co-evolution to win over simply using a larger training set, it must either be faster or the curve must go lower than whatever line corresponds to the amount of data which must have been processed to reach this distribution. Unfortunately we cannot answer this question without putting in dynamics, and it is a crucial question.

Figure 2 shows the experimental results, averaged over t for larger values of β. The beginning of the transition to the oscillating state can be seen. If continued on for higher values of β this would go to 1/2. There is an optimal value of β which leads to an optimal true performance. Beyond this value, the oscillatory state emerges. This value grows like $\log(N_d)$.

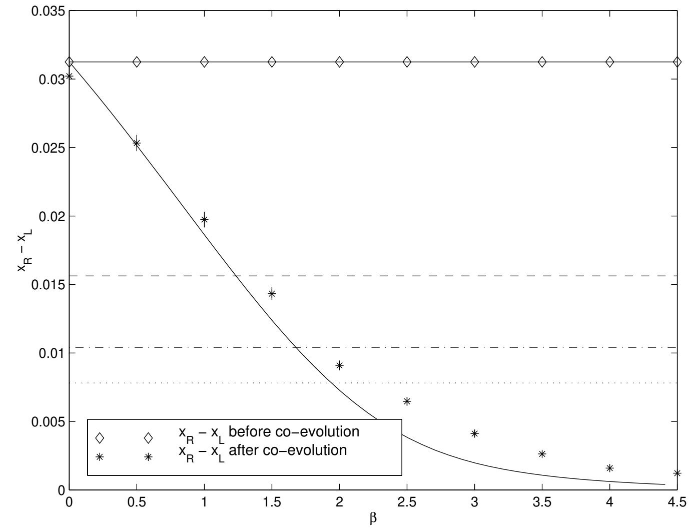

Fig. 1. The true performance of the models after a co-evolution step versus that before. The solid lines are theory; the horizontal line is before co-evolution. The line after co-evolution goes to zero showing that co-evolution can lead to arbitrary good generalization with suitably chosen parameters and infinite population size. The diamonds and asterisks are experiments using direct sampling from the Gibbs distribution. The other horizontal lines show what the true performance would without co-evolution but with the size of the data set doubled (dashed line), tripled (dot-dashed line), and quadrupled (dotted line). The parameters are: $N_s = 50$, $N_d = 64$, $t = 0.42$.

5 Conclusions

Whether data-model co-evolution can work by evolving the data to a set that leads to improved generalization was considered. It cannot be proven to be so, it was argued. A very simple toy example was solved analytically. It was shown that this system is unstable to oscillations between very low fitness states. If the learning parameter is suitably scaled, however, the data could evolve to a set leading to improved generalization performance. If continued for larger β, it would go to 1/2.

5.1 Future work

In order to understand data-model co-evolution, much more work is required. First, we must put in dynamics, in order to estimate the time to get to this state. This is essential in calculating the efficiency of the approach. This could be done for the simulated annealer, although GA dynamics would be quite relevant. This problem is quite trivial; it would be important to find more interesting problems

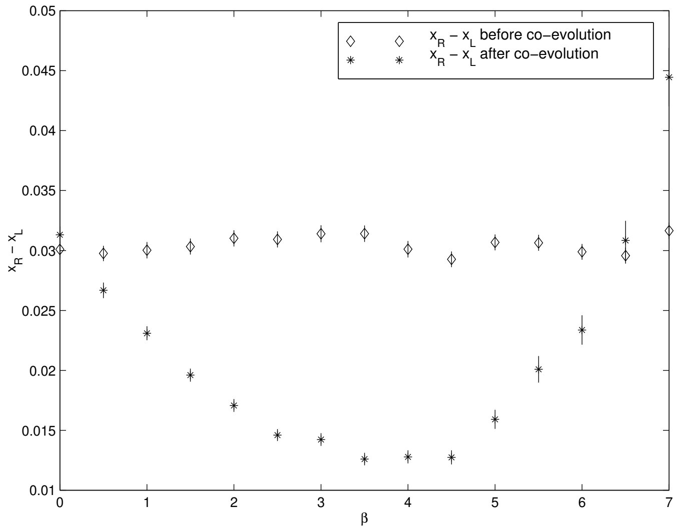

Fig. 2. The experiments of figure 1 continued for higher values of β and averaged over t. The beginning of the transition to the oscillatory behavior is seen; this is a finite population effect.

to analyse. It also might be interesting to attempt to put in interactions between the individuals.

To study the shaping hypothesis, it will be necessary to work out the dynamics in full detail, including getting the learning time. This is difficult for GAs without co-evolution, but it is clearly a very interesting problem.

References

1. Hillis, W.D. *Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure*, Physica D **42**, 228-234 (1990).
2. Paredis, Jan, *Coevolutionary Algorithms* from the Handbook of Evolutionary Computation, Bäck, T. Fogel, D. Michalewicz Z. (eds.) Oxford University Press, 1997.
3. Paredis, Jan,*Coevolving Cellular Automata: Beware of the Red Queen!*, Proceedings of the Seventh Conference on Genetic Algorithms, Bäck, T. editor, 393–399, 1997.
4. Rattray,M *Modelling the Dynamics of Genetic Algorithms Using Statistical Mechanics*, University of Manchester Ph.D. thesis, 1996.
5. Rosin, C.D., *Coevolutionary Search Among Adversaries*, University of California Ph.D. thesis, 1997.
6. Valiant, L.G. *A theory of the learnable*, Communications of the ACM, **27** no. 11, 1134 – 1142, 1984.