Accelerating Checkers AI Evolution

Karl Cobbe
Stanford University
kcobbe@stanford.edu

Paul Lee
Stanford University
pslee@cs.stanford.edu

Andres Gomez-Emilsson
Stanford University
nc07agom@stanford.edu

ABSTRACT

This project aims to determine the extent to which evolutionary algorithms can be enhanced by extrapolating local evolutionary trends in the state space of possible agents. By evaluating the fitness of many clustered agents, evolutionary algorithms generate a significant quantity of information about the local structure of the state space. Basic implementations of evolutionary algorithms, however, do not fully utilize this information. In particular, mutations for each generation are random; they ignore any structure in the state space. Conventional evolution advances solely based on selecting the fittest agents from generation to generation. We propose to learn structure in the state space, through SVM regression, and to favor those mutations which perform well, as estimated by the learned model. We call this method SVM enhanced evolution. We show that when used to evolve AI agents to play Checkers, SVM enhanced evolution consistently out-performs basic implementations of evolutionary algorithms in both noisy and non-noisy conditions. By extrapolating local evolutionary trends, SVM enhanced evolution achieves locally optimal performance in approximately a third as many generations as conventional evolution.

1. INTRODUCTION

To provide the framework for testing SVM enhanced evolution, we chose to evolve neural networks to play Checkers. We repeatedly conducted evolution on a fixed set of agents, measuring the rates at which performance improves. We computed both the average rate of performance improvement in conventional evolution and the average rate of performance improvement in SVM enhanced evolution. This was the metric used to compare the two algorithms.

In the case of playing Checkers, as well as many other evolutionary problems, the fitness evaluation function is costly. To get a reliable measure of the performance of any given agent, hundreds of games of Checkers must be played. Applying SVM regression to estimate local fitness levels in the agent state space is orders of magnitude less computationally intensive than conventional evaluations. Since playing through an entire game of Checkers is a relatively complex computation, bypassing actual game play can significantly improve the rate of evolution. In this way, SVM enhanced evolution benefits from the randomness of conventional evolution, as well as the intelligence and guidance of machine learning.

2. PRIOR WORK

The idea of evolving neural networks to play Checkers is based on the success Chellapilla and Fogel had in evolving their own Checkers neural networks (in 1999, on far less sophisticated hardware). They did not attempt to use machine learning to speed up the evolution.

3. METHOD

We will begin by describing the conventional evolutionary algorithm, and then we will describe how we implemented SVM regression to improve conventional evolution.

3.1 Implementation

Each neural network has 32 inputs (for each playable square on the Checkers board). The inputs are 1 for a red piece, -1 for a black piece, 1.5 for a red king, -1.5 for a black king, and 0 for empty. Valuing kings and checkers in a 3:2 ratio is known to be a decent approximation. A high positive value indicates a board position favorable to red, and a high negative value indicates a board position favorable to black. Aside from the second hidden layer, there is a single additional input to the output neuron, namely, the piece count difference of the inputted board (number of black pieces subtracted from the number of red pieces, with kings appropriately weighted). The weight of this piece difference input, like all other weights in the neural network, is an evolvable parameter.

Assigning an estimated value to board positions is the bulk of our AI strategy. To actually select a move, the AI performs a standard minimax search of the game tree down to a ply depth of 4. Since such a limited depth does not capture much of the game space, it is clear that the ability of the neural network to value board positions becomes important for our AI to succeed.

3.2 Conventional Evolution

To test the effectiveness of conventional evolution, we began by evolving a neural network with 2 hidden layers of 40 neurons and 10 neurons. This was following the procedure used by Chellapilla and Fogel. Using these relatively large

Figure 1: The figure on the left depicts the exploration of the fitness landscape performed by a single evolutionary iteration. It highlights and makes use of only a fraction of the information available, namely, the sections that score higher than a set threshold in the evaluation function for a particular region of the state space. The figure on the right depicts the approach explored in this project, where the historical evaluations are taken into account to infer the underlying structure of the fitness landscape.

neural networks, we were able to obtain similar results, as coarsely evaluated against online Checkers programs.

To implement SVM enhanced evolution, we chose to significantly reduce the number of dimensions in the agent state space. By making each agent a single layer neural network, in which each of the 32 inputs directly feeds into the output, we can uniquely describe each agent in 33 dimensions (32 dimensions for the input weights, plus a weight for the piece differential). All comparisons between conventional evolution and SVM enhanced evolution were made using these single layer neural networks.

Each generation is seeded by 50 agents. Each agent produces a single child, so there are 100 agents in all, per generation. Each agent then plays NUMGAMES games (half as red, half as black) against a standard minimax strategy. For reference, a standard minimax strategy can play competent, mediocre Checkers. Each neural net receives +1,0, or -1 points for a win, draw, or loss, respectively. The 50 most successful agents become the center of the subsequent generation, by way of mutation.

Evaluating an agent can be expensive, as each game played is comparatively computationally expensive. We must therefore tradeoff between speed and confidence in our evaluations. In this way, NUMGAMES is an important parameter to select. Averaging over relatively few games yields evaluations with high variance. Averaging over many games yields evaluations with low variance but at the cost of computational resources.

In addition to NUMGAMES, many other important evolutionary parameters must be selected to govern the mechanics of each generation. In particular, the following questions must be answered: How large should each generation be? How many survivors should pass their genes onto the next generation? These questions are difficult to answer, and different answers often yield markedly different results. If we allow too few survivors or perform too coarse evaluations, information is easily lost through generations. If we allow too many survivors or perform too thorough evaluations, computational resources are needlessly wasted. We hardcoded these parameters based on empirically observing which parameters led to high performing evolutionary runs. Determining a generalized optimal answer is beyond the scope of this paper, but it is nevertheless an important consideration.

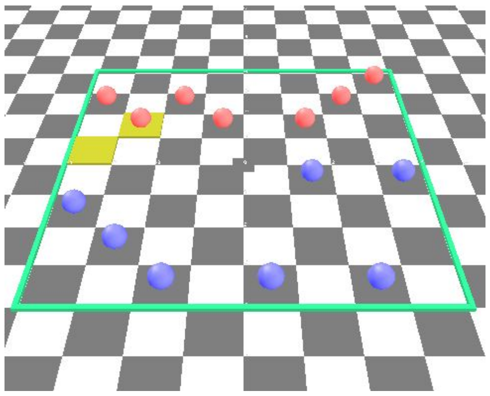

Figure 2: The figure above depicts our Checkers graphical user interface.

3.3 SVM Enhanced Evolution

SVM enhanced evolution requires the following modifications to conventional evolution. The performance of each agent in a given generation is recorded, and this information is stored even beyond the lifetime of the agent. At the start of each generation, SVM regression is performed on the dataset of past agent performance. This generates a model of the agent state space. A large number of children are then generated through mutation. The 100 best children, as judged by the learned model, are selected, and conventional evolution resumes.

Simply put, SVM enhanced evolution provides a layer of filtering on the mutations of each generation. Only promising mutations are evaluated; the rest are discarded. Because evaluation by the learned model requires negligible computation, as compared to conventional evaluation, SVM enhanced evolution can consider a much larger number of children during each generation.

3.4 Parameters

With the use of cross-validation we found that the radial basis function was the most successful kernel type. In every run, cross-validation over several different parameter sets determined the appropriate cost and gamma parameters for this kernelization.

There are two important parameters unique to SVM enhanced evolution. The first is TAILLENGTH, and the second is MAXJUMP.

TAILLENGTH determines how many data points are considered when performing SVM regression. In particular, the most recent TAILLENGTH data points are used. As more data points are gathered, the oldest data points are discarded. This process tracks the path evolution has traversed through the state space, called the tail.

The second important parameter unique to SVM enhanced evolution is MAXJUMP. MAXJUMP determines the maximum Euclidean distance permitted between a parent and a child, when selecting that child based on the results of SVM regression.

We will discuss the nature of these parameters, how we chose to select their values, and how poor parameter selection impacts evolution.

4. RESULTS AND DISCUSSION

We tested SVM enhanced evolution against conventional evolution on several different parameter sets. Two of these sets are shown in Figures 3 and 4. These graphs each contrast 6 generation runs, 3 of which use SVM enhanced evolution and 3 of which use conventional evolution. SVM enhanced evolution is colored orange, and conventional evolution is colored purple. The generation runs in Figure 3 use a relatively conservative parameter sets. The generations are large; there are a relatively high number of survivors from generation to generation; and, each agent is thoroughly evaluated before any comparisons are made. In contrast, Figure 4 is a much noisier data set. Generations are smaller; few agents survive between generations; and, only a handful of games are played between agents before selecting the winners.

Naturally, the generations in Figure 4 can be evaluated faster than the generations in Figure 3. However, improving performance level takes more generations in the noisier runs. This is the essence of the tradeoff between thorough and coarse evaluation.

One way of quantitatively assessing the relative speed at which the strategies perform is to compare the average number of generations required to achieve a certain performance level. Table 1 summarizes the number of generations re-

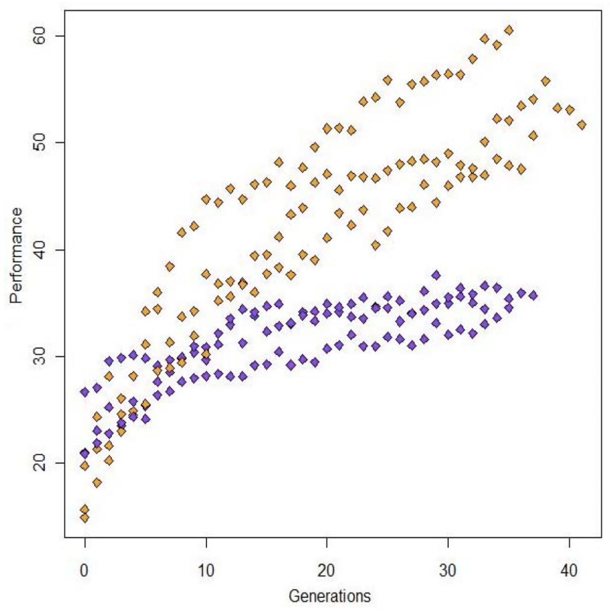

Figure 3: Regular Evolution vs. SVM Enhanced Evolution on a non-noisy data set

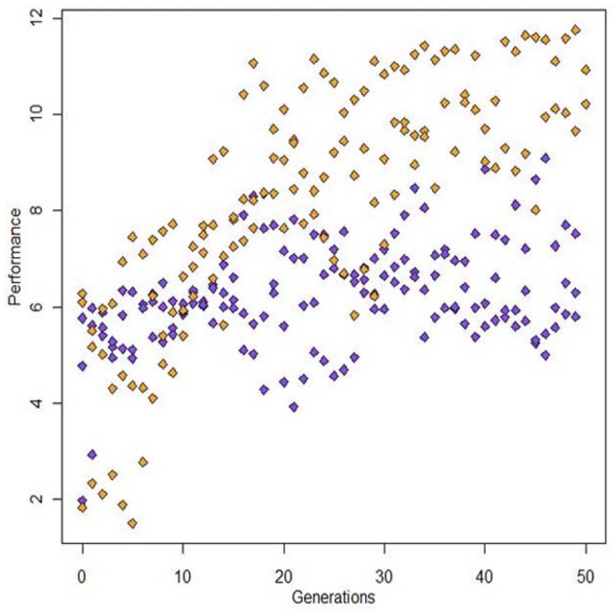

Figure 4: Regular Evolution vs. SVM Enhanced Evolution on a noisy data set

quired to achieve approximately 70% of asymptotic performance. The parameter NUMGAMES in the table corresponds to the number of games that are used to asses the performance of the agents. When this number is small, the evaluation function is less precise and the data points are therefore more noisy. In both conditions, SVM enhanced evolution outperforms regular evolution by an average factor of 3. Even in noisy conditions SVM regression was able to illuminate underlying structure of the local fitness landscape.

It is interesting to examine the cause for the performance differential between SVM enhanced evolution and conventional evolution in the noisy runs in Figure 4. In these runs, the evaluation of any given generation contains relatively unreliable data. In conventional evolution, the algorithm's only memory lies in the genes of the latest generation. In contrast, SVM enhanced evolution maintains memory not only in this generation, but also in the dataset of the performance of past generations. It is on this dataset that SVM regression is performed to obtain an estimated model of the state space. This increase in relevant memory enables SVM evolution to learn from generations that conventional evolution has long forgotten. Against coarse generation evaluations, this proves to be a significant advantage.

Table 1: Average Number of Generations to Score of 10 by Type of Evolution

Noise Level	SVM Enhanced	Regular
1	83	264
5	33	106
10	23	55
50	7.7	20.7
100	4.8	13.1

Either overvaluing or undervaluing TAILLENGTH negatively impacts SVM enhanced evolution. If too few data points are in the tail, then SVM regression is unable to estimate an accurate model of the state space. In this case, any extrapolation done based on the results of SVM regression is only likely to hinder evolution. Similarly, if too many data points are used, then SVM regression might be operating on a largely stale dataset. Long obsolete genotypes will inhibit SVM regression's ability to analyze more recent trends. As before, extrapolation based on such results will likely hinder evolution.

Regarding the parameters, we chose to set TAILLENGTH to be 600. We are working in a 33 dimensional space, and we figured that 20 data points per dimension was a reasonable starting point. After conducting several trials with TAILLENGTH ranging from 100 to 2000, we empirically observed 600 to be a reasonable choice.

We found that if the parameter, MAXJUMP, is too low, then SVM regression will have a negligible impact on conventional evolution. If MAXJUMP is too high, then SVM regression will extrapolate too far, overestimating the performance of many child agents, thereby steering evolution off course.

These results were again confirmed by empirical observation; excessively low or high values of MAXJUMP yielded poor results. We selected an intermediate value for MAXJUMP, which we observed performed well.

5. TOPICS FOR FURTHER RESEARCH

Further research can focus on two general directions: optimizing the current method, and exploring further applications of SVM enhanced evolution.

For the purpose of improving our current method, the parameter selection of the evolutionary algorithm can be optimized. The process of parameter selection could be automated so that parameters are updated from generation to generation, adapting as needed, based on feedback from generation performance. This could significantly improve the overall performance of SVM enhanced evolution. Furthermore, it might be worth exploring different kernelizations. The current approach uses a radial basis kernel, however, a comprehensive study of the structure of the state space might show that other kernel types are more appropriate.

A possible way to improve the precision of the evaluation function would be to use a tournament rating system such as the Elo algorithm. By doing this, the necessary number of games played to assess the fitness of the different agents within a generation could be reduced without compromising the accuracy of the evaluation. The Elo rating system could cope with the statistical variability in game outcomes much better than simple averaging. In this way, a reliable rating for an untested agent could be obtained with fewer games.

With regards to the applicability and generality of SVM enhanced evolution, it is worth investigating how well this method scales to higher dimensions. It is possible to argue that most state spaces have an underlying structure, but high dimensionality might make the task of detecting that structure intractable. We showed that the radial basis kernelization can detect some useful underlying structure of the fitness of the state space of neural networks, but does this generalize to the state spaces of other types of strategies, possibly represented in higher dimensions?

6. BIBLIOGRAPHY

Fogel, David B. *Blondie24: Playing at the Edge of AI*. San Francisco: Morgan Kaufmann, 2002. Print.

Fogel, David B. "Evolving a Checkers Player without Relying on Human Experience." *Intelligence* 11.2 (2000): 20-27. Web.