Evolving Soccer Strategies

Rafał Sałustowicz, Marco Wiering, Jürgen Schmidhuber

IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland
e-mail: {rafal, marco, juergen}@idsia.ch

In N. Kasabov, R. Kozma, K. Ko, R. O'Shea, G. Coghill, and T. Gedeon, editors, Progress in Connectionist-based Information Systems: Proceedings of the Fourth International Conference on Neural Information Processing ICONIP'97, volume 1, pages 502-505. Springer-Verlag Singapore, 1997.

Abstract

We study multiagent learning in a simulated soccer scenario. Players from the same team share a common policy for mapping inputs to actions. They get rewarded or punished collectively in case of goals. For varying team sizes we compare the following learning algorithms: TD-Q learning with linear neural networks (TD-Q-LIN), with a neural gas network (TD-Q-NG), Probabilistic Incremental Program Evolution (PIPE), and a PIPE variant based on coevolution (CO-PIPE). TD-Q-LIN and TD-Q-NG try to learn evaluation functions (EFs) mapping input/action pairs to expected reward. PIPE and CO-PIPE search policy space directly. They use adaptive probability distributions to synthesize programs that calculate action probabilities from current inputs. We find that learning appropriate EFs is hard for both EF-based approaches. Direct search in policy space discovers more reliable policies and is faster.

1 Introduction

There are at least two classes of candidate algorithms for multiagent reinforcement learning (RL). The first includes traditional single-agent RL algorithms based on adaptive evaluation functions (EFs) [Bertsekas, 1996]. Usually online variants of dynamic programming and function approximators are combined to model EFs mapping input-action pairs to expected discounted future reward. EFs are then exploited to select actions. Methods from the second class do not require EFs. Their policy space consists of complete algorithms defining agent behaviors, and they search policy space directly. Members of this class are Levin search [Levin, 1973], Genetic Programming, e.g., [Cramer, 1985], and Probabilistic Incremental Program Evolution (PIPE) [Sałustowicz and Schmidhuber, 1997].

Previous Results. Recently we compared two learning algorithms [Sałustowicz *et al.*, 1997b], each representative of its class: TD-Q learning [Lin, 1993] with linear neural nets (TD-Q-LIN) and PIPE. We let both approaches compete against a biased random opponent (*BRO*). PIPE quickly learned to beat *BRO*. TD-Q-LIN had difficulties in learning appropriate shared EFs, especially in case of multiple agents per team.

Comparisons. The current paper extends our previous work in several ways: *(1)* Since TD-Q-LIN's EF approximation capabilities are limited we combine TD-Q with a more powerful function approximator: the neural gas network (TD-Q-NG). *(2)* Since good hand-made opponents are not always easy to design, we test whether PIPE can coevolve good programs by letting them play against each other instead of *BRO* (CO-PIPE).

2 Soccer Simulation

There are either 1 or 11 players per team. Players can move with and without the ball or shoot it. As in indoor soccer the field is surrounded by impassable walls except for the two goals centered in the east and west walls. The ball slows down due to friction (after having been shot) and bounces off walls obeying the law of equal reflection angles (we simulate in discrete time). Players are "solid". If a player, coming from a certain angle, attempts to traverse a wall then it "glides" on it, losing only that component of its speed which corresponds to the movement direction hampered by the wall. Collisions of players cause them to bounce back to their positions at the previous time step. If one of them had the ball then the ball changes owners. There are fixed initial positions for all players and the ball (see Figure 1). A game lasts from time $t = 0$ to time t_{end}.

Action Framework/Cycles. At each discrete time step $0 \leq t < t_{end}$ each player executes a "cycle". A cycle consists of: (1) an attempt to get the ball, if it is close enough, (2) input computation, (3) action selection and execution, and (4) another attempt to get the ball, if it is close enough. Once all players have executed a cycle we move the ball. If a team scores or $t = t_{end}$ then all players and ball are reset to their initial positions.

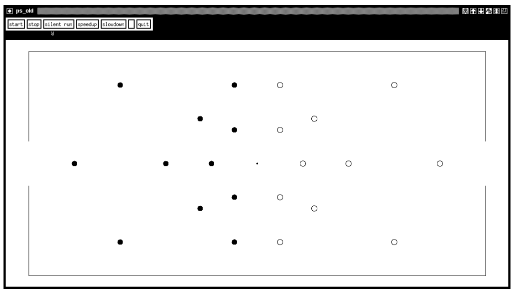

Fig. 1: 22 players and ball in initial positions. Players of 1 player teams are the goalkeepers in the back.

Inputs. Player p's input at a given time t is an input vector $\vec{i}(p,t)$. Vector $\vec{i}(p,t)$ has 14 components: (1) Three boolean inputs that tell whether the player/a team member/an opponent has the ball. (2) Polar coordinates (distance, angle) of both goals and the ball with respect to a player-centered coordinate system. (3) Polar coordinates of both goals with respect to a ball-centered coordinate system. (4) Ball speed. Note that these inputs do not make the environment fully observable.

Actions. Players may execute actions from action set *ASET*. *ASET* contains: *go_forward*, *turn_to_ball*, *turn_to_goal* and *shoot*. Shots are noisy and noise makes long shots less precise than close passes. For a detailed description of the simulator see [Sałustowicz *et al.*, 1997a].

3 Probabilistic Incremental Program Evolution (PIPE)

PIPE [Sałustowicz and Schmidhuber, 1997] synthesizes programs which select actions from $ASET$, given player p's input vector $\vec{i}(p,t)$.

Action Selection. Action selection depends on 5 variables: $g \in I\!R$, $A_i \in I\!R$, $\forall i \in ASET$. Action $i \in ASET$ is selected with probability P_{A_i} according to the Boltzmann distribution at temperature $\frac{1}{g}$:

$$P_{A_i} := \frac{e^{A_i \cdot g}}{\sum_{j \in ASET} e^{A_j \cdot g}} \qquad \forall i \in ASET \tag{1}$$

All A_i and g are calculated by a program.

Programs. A main program PROGRAM consists of a program PROG^g which computes the "greediness" parameter g and 4 "action programs" PROG^i ($i \in ASET$). The result of applying PROG to data x is denoted $\text{PROG}(x)$. Given $\vec{i}(p,t)$, $\text{PROG}^i(\vec{i}(p,t))$ returns A_i and $g := |\text{PROG}^g(\vec{i}(p,t))|$. An action $i \in ASET$ is then selected according to (1).

Program Instructions. A program PROG contains instructions from a function set F and a terminal set T. We use $F = \{+, -, *, \%, sin, cos, exp, rlog\}$ and $T = \{\vec{i}(p,t)_1, \ldots, \vec{i}(p,t)_v, R\}$, where % denotes protected division ($\forall y, z \in I\!R, z \neq 0$: $y\%z = y/z$ and $y\%0 = 1$), *rlog* denotes protected logarithm ($\forall y \in I\!R, y \neq 0$: $rlog(y)$=log(abs(y)) and $rlog(0) = 0$), $\vec{i}(p,t)_l$ $1 \leq l \leq v$ denotes component l of a vector $\vec{i}(p,t)$ with v components and R represents the *generic random constant* from [0;1).

PIPE Overview. PIPE programs are encoded in n-ary trees that are parsed depth first from left to right, with n being the maximal number of function arguments. PIPE generates programs according to a probability distribution over all possible programs composable from the instruction set ($F \cup T$). The probability distribution is stored in an underlying *probabilistic prototype tree* (*PPT*). The *PPT* contains at each node a probability for each instruction from $F \cup T$ and a random constant from [0;1). Programs are generated by traversing the *PPT* depth first from left to right starting at the root node. At each node an instruction is picked according to the node's probability distribution. In case the *generic random constant* is picked it is instantiated either to the value stored in the *PPT* node or a random value from [0;1), depending on the instruction's probability. To adapt *PPT*'s probabilities PIPE generates successive populations of programs. It evaluates each program of a population and assigns it a scalar, non-negative "fitness value", which reflects the program's performance. To evaluate a program we play one entire soccer game against a hand-made biased random opponent and define the program's fitness to be: *100 - number of goals scored by learner + number of goals scored by opponent.* The offset 100 is sufficient to ensure a positive score difference. PIPE then adapts *PPT*'s probabilities so that the probability of creating the best program of the current population increases. Finally *PPT*'s probabilities are mutated to better explore the search space. All details can be found in [Sałustowicz and Schmidhuber, 1997].

Coevolution (CO-PIPE). CO-PIPE works exactly like PIPE, except that: (a) the population contains only two programs and (b) we let both programs play against each other rather than against a prewired opponent. CO-PIPE adapts *PPT*'s probabilities so that the probability of creating the winning program increases.

4 TD-Q Learning

In a previous paper [Sałustowicz *et al.*, 1997b] we found that learning correct soccer EFs was hard for an offline TD(λ) Q-variant [Lin, 1993] with linear neural nets. Here we use a neural gas network instead [Fritzke, 1995]. The goal is to map a player-specific input $\vec{i}(p,t)$ to action evaluations $Q(\vec{i}(p,t), a_d)$, where $a_d \in ASET$. We use the same neural gas network for all policy-sharing players. We reward the players equally whenever a goal has been made or the game is over.

Action Selection. We use a set of Z neurons: $\{n_1, \ldots, n_Z\}$ (initially $Z = Z_{init}$). They are placed in the input space by assigning to each a location $\vec{w}_k \in I\!R^{14}$ (with $\vec{i}(p,t)$'s dimension). $\forall k \in \{1, \ldots, Z\}$, n_k contains a

Q-value $Q_k(a_d)$ for each $a_d \in ASET$. To select an action we calculate overall Q-values by combining Q-values of all neurons. First we calculate a weighting factor g_k for each neuron n_k:

$$g_k := \frac{e^{-\eta \; dist(\vec{w}_k, \vec{i}(p,t))}}{\sum_{j=1}^{Z} e^{-\eta \; dist(\vec{w}_j, \vec{i}(p,t))}},$$

where $dist(\vec{w}_k, \vec{i}(p,t))$ is the Manhattan distance between player input and the location of neuron n_k, and $\eta \in I\!R$ is a user-defined constant. The overall Q-value of an action a_d, given input $\vec{i}(p,t)$, is

$$Q(\vec{i}(p,t), a_d) := \sum_{j=1}^{Z} g_j Q_j(a_d)$$

Once all Q-values have been calculated, a single action is chosen according to the Max-Random rule: select the action with highest Q-value with probability P_{max}, otherwise select a random action.

TD-Q Learning. Each game consists of separate trials. A given trial stops at time t^* once one of the teams scores or the game is over ($t^* = t_{end}$). To achieve an optimal strategy we want the Q-value $Q(\vec{i}(p,t), a_d)$ for selecting action a_d given input $\vec{i}(p,t)$ to approximate

$$Q(\vec{i}(p,t), a_d) \sim \mathcal{E}(\gamma^{t^*-t} R(t^*)),$$

where $\mathcal{E}$ denotes the expectation operator, $0 \le \gamma \le 1$ the discount factor which encourages quick goals (or a lasting defense against opponent goals), and $R(t^*)$ the reinforcement at trial end (-1 if opponent team scores, 1 if own team scores, 0 otherwise).

To learn these Q-values we monitor player experiences (inputs and selected actions) in player-dependent history lists with maximum size H_{max}. After each trial we calculate examples using the TD-Q method. For each player history list, we compute desired Q-values $Q^{new}(p,t)$ for selecting action a_d, given $\vec{i}(p,t)$ $(t = t^1, \ldots, t^*)$ as follows:

$$\begin{aligned} Q^{new}(p,t^*) &:= R(t^*); \\ Q^{new}(p,t) &:= \gamma \cdot [\lambda \cdot Q^{new}(p,t+1) + (1-\lambda) \cdot \\ & \quad Max_d\{Q(\vec{i}(p,t+1), a_d)\}] \quad \forall t \neq t^*. \end{aligned}$$

λ determines subsequent experiences' degree of influence.

Learning Rules. There are two goals: (1) learning network structure — move the neurons to locations where they help to minimize overall error, and (2) learning Q-values — make individual neurons correctly evaluate the inputs for which they are used.

For learning a specific example $(\vec{i}(p,t), a_d, Q^{new}(p,t))$, we introduce for each neuron n_k a responsibility variable which is adapted at each cycle: $C_k := C_k + g_k$.

(1) Learning Structure. If the error $|Q(\vec{i}(p,t), a_d) - Q^{new}(p,t)|$ of the system is larger than an error-threshold T_E, the number of neurons is less than Z_{max}, and the closest neuron's responsibility C_k exceeds the responsibility threshold T_C, then we add a new neuron n_{Z+1}. We set its location $\vec{w}_{Z+1}$ to $\vec{i}(p,t)$, copy all Q-values from the closest neuron to the new neuron except for the Q-value of action a_d which is set to the desired Q-value $Q^{new}(p,t)$. Finally we set $Z := Z + 1$.

If no neuron is added, we calculate for each neuron n_k ($\forall k \in \{1, \ldots, Z\}$) a gate-value h_k, which reflects the posterior belief that neuron n_k evaluates the input best:

$$h_k := \frac{g_k e^{-(Q^{new}(p,t) - Q_k(a_d))^2}}{\sum_{j=1}^{Z} g_j e^{-(Q^{new}(p,t) - Q_j(a_d))^2}}$$

We then move each neuron n_k towards the example $\vec{i}(p,t)$ according to h_k:

$$\vec{w}_k := \vec{w}_k + lr_k h_k^2 (\vec{i}(p,t) - \vec{w}_k),$$

where $lr_k := lr_N (C_k)^{-\beta}$, lr_N is the system learning rate and β is the learning rate decay factor.

(2) Learning Q-values. Each neuron k's Q-value for selecting action a_d is updated as follows:

$$Q_k(a_d) := Q_k(a_d) + lr_k h_k (Q^{new}(p,t) - Q_k(a_d))$$

5 Experiments

For each combination of learning algorithm (TD-Q-LIN, TD-Q-NG, PIPE, and CO-PIPE) and team size (1 and 11) we perform 10 independent runs, each comprising 3300 games of length $t_{end} = 5000$. Every 100 games we test current performance by playing 20 test games (no learning) against a biased random opponent *BRO* and summing the score results. *BRO* randomly executes actions from *ASET*. *BRO* is not bad due to the initial bias in the action set. If we let *BRO* play against a non-acting opponent *NO* (all *NO* can do is block) for twenty 5000 time step games then *BRO* wins against *NO* with on average 71.5 to 0.0 goals for team size 1 and 108.6 to 0.5 goals for team size 11.

PIPE and CO-PIPE Set-ups. Parameters for PIPE runs are: P_T=0.8, $\varepsilon = 1$, $P_{el} = 0$, PS=10, lr=0.2, P_M=0.1, mr=0.2, T_R=0.3, T_P=0.999999 (see [Sałustowicz and Schmidhuber, 1997] for details). For CO-PIPE we keep the same parameters except for PS, which is set to 2 (see Section 3). During performance evaluations we test the best-of-current-population program (except for the first evaluation where we test a random program).

TD-Q-LIN and TD-Q-NG Set-ups. After a thorough parameter search we found the following best parameters for TD-Q-LIN runs: γ=0.99, Lr^N=0.0001, λ=0.9, H_{max}=100. Weights are randomly initialized in $[-0.01, 0.01]$. For TD-Q-NG we used: γ=0.98, lr_N=0.1, λ=0.9, H_{max}=100, $\beta = 0.1$, $\eta = 30$, Z_{init}=10, Z_{max}=100, $P_{max} = 0.7$, T_E=0.5, $T_C = 1000$. $\vec{w}_k$ components are randomly initialized in $[-1.0, 1.0]$, Q-values are zero-initialized.

Results. We plot goals scored by learner and opponent during test phases against number of games in Figure 2. PIPE's score differences continually increase. It always quickly learns an appropriate policy regardless of team size. CO-PIPE also finds successful policies. Its score differences are smaller than PIPE's. This, however, is an expected outcome since CO-PIPE never met *BRO* during training. CO-PIPE's performance increases with increasing team size, since it becomes easier to distinguish between good and bad policies. PIPE and CO-PIPE achieve much better performance than TD-Q-LIN and TD-Q-NG. This is partially due to PIPE's and CO-PIPE's ability to efficiently select relevant input features for each action. TD-Q-LIN's score differences first increase until TD-Q-LIN runs into an "outlier problem", which lets its linear nets unlearn previously discovered good policies (see [Sałustowicz *et al.*, 1997b] for details). TD-Q-NG initially learns faster than TD-Q-LIN, but does not continue improving. It stays quite stochastic during the entire run.

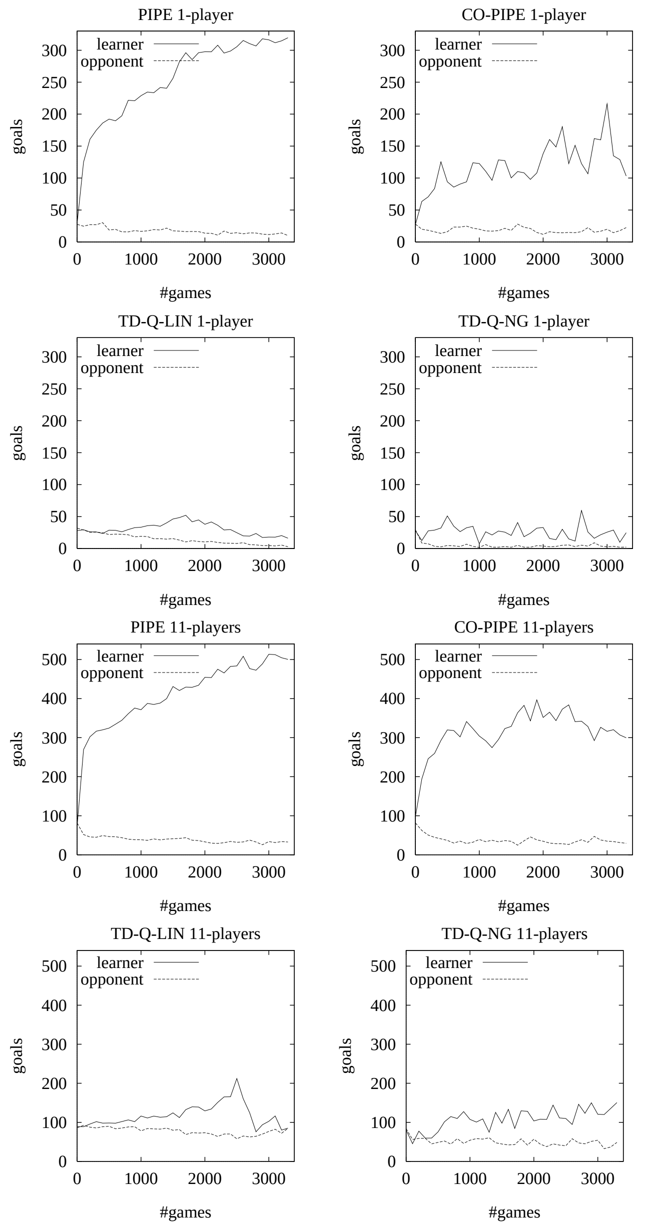

Fig. 2: Average number of goals scored during all test phases, for team sizes 1 and 11.

6 Conclusion

We compared two direct policy search methods (PIPE and CO-PIPE) and two EF-based ones (TD-Q-LIN and TD-Q-NG) in a simulated soccer case study with policy-sharing agents. PIPE, TD-Q-LIN, and TD-Q-NG were trained against a biased random opponent (*BRO*). CO-PIPE evolved its policies by coevolution. PIPE and CO-PIPE quickly learned to beat *BRO*, CO-PIPE even *without* being explicitly trained to do so. TD-Q-LIN and TD-Q-NG achieved performance improvements, too. Despite our efforts to improve EF-based approaches by using different function approximators (linear nets and the more powerful neural gas nets) their results remain less exciting. TD-Q-LIN's and TD-Q-NG's problems are due to difficulties in learning EFs in partially observable stochastic environments.

Acknowledgments

Thanks for valuable discussions to Jieyu Zhao, Nic Schraudolph, Luca Gambardella, and Cristina Versino.

References

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

[Cramer, 1985] Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J., editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Hillsdale NJ. Lawrence Erlbaum Associates.

[Fritzke, 1995] Fritzke, B. (1995). A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA.

[Levin, 1973] Levin, L. A. (1973). Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266.

[Lin, 1993] Lin, L. J. (1993). *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Carnegie Mellon University, Pittsburgh.

[Sałustowicz and Schmidhuber, 1997] Sałustowicz, R. P. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2).

[Sałustowicz et al., 1997a] Sałustowicz, R. P., Wiering, M. A., and Schmidhuber, J. (1997a). Learning team strategies with multiple policy-sharing agents: A soccer case study. Technical Report IDSIA-29-97, IDSIA.

[Sałustowicz et al., 1997b] Sałustowicz, R. S., Wiering, M. A., and Schmidhuber, J. (1997b). On learning soccer strategies. In *Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN'97), Lecture Notes in Computer Science.* Springer-Verlag Berlin Heidelberg. To appear.