Planning using online evolutionary overfitting

Spyridon Samothrakis and Simon M. Lucas, *Senior Member, IEEE*

***Abstract*— Biological systems tend to perform a range of tasks of extreme variability with extraordinary efficiency. It has been argued that a plausible scenario for achieving such versatility is explicitly learning a forward model. We perform a set of experiments using the original and a modified version of a classic reinforcement learning task, the mountain car problem, using a number of agents that encode both a direct and an abstracted version of a forward model. The results suggest that superior performance can be achieved if the forward model can be exploited in real-time by an agent that has already internalised a model-free control function.**

I. Introduction

The idea that the brain encodes a model of reality and uses that model to reason about the world has a long history in almost all fields of science that relate to action-selection and animal/agent behaviour.

In classical Artificial intelligence (A.I.), although never explicitly stated, a model of the world is used to infer future rewards (ex. Min-max, see [1]) based on some tree-based search method. The power of classical A.I. to find near optimal solutions given a model (and enough computational power) has recently been exposed in a popular game competition[2], where the winning agents encoded a perfect model of reality (the Mario world in this case) and used A^* [3] to reason using that model.

More recently, theories of brain function put forward by Friston et al.[4], [5] require that the brain encodes a generative encoding of its sensory input, as an indispensable part of its function. The generative model complements the brain's recognition model, and all action stems from the fact that an agent tries to reconcile its internal world model with reality, i.e. minimise its prediction error.

Another successful effort of sensory input modelling has been Dynamic Adaptive Control(DAC)[6]. DAC was partially created to show how behaviour can emerge from the interaction of different cortical areas. It defines a generic multi-component agent architecture, with one of the components responsible for predicting future sensory inputs. In a hybrid wall avoidance/phototaxis task, it was shown that agents that specifically try to predict their future manage to minimise the entropy of their sensory input, i.e. have more stable trajectories.

A more abstract line of thinking concerning the idea of encoding forward models comes from cybernetics[7] and neural networks[8]. The ideas in these papers predate DAC and the free energy formulation, and expose the theoretical benefits of a forward model.

Spyridon Samothrakis and Simon M. Lucas are with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom. (emails: ssamot@essex.ac.uk, sml@essex.ac.uk).

From the field of reinforcement learning, a number of model based systems have been proposed, most notably Dyna-Q[9] and Dyna-Q2[10]. The approach followed in these papers has more to do with using the model to do planning than to correct possible corrupted input.

In Evolutionary biology, it has been argued that a possible reason behind the development of the mind was the ability of its carrier to perform mental simulations of possible future events, thus allowing the agent to "test" its theories about the future[11] without physical harm.

From adaptive behaviour, there is a number of publications (ex. [12], [13]) that exploit the model as part of a setup called "anticipatory" behaviour, that are strongly focused on using internal planning to guide the actions for an agent.

Finally, from a neuroscientific perspective, all the above uses of a forward model[1] are discussed in [15]. For the paper's main experiment, a number of participants where asked to move their hands in total darkness and asess the new position. The experimental data collected can be accounted for by using a kalman filter[16]. This supports the view that indeed a forward model is encoded by the Central Nervous System, as a kalman filter requires such a model.

In this paper we revisit the subject of sensory/state input modelling in a somewhat ad-hoc manner, using methods from both Artificial life and classical machine learning to create an agent that reasons using a model of its environment.

We claim three contributions. First, we show that one can use a perfect forward model in order to boost the performance of an agent after reactive/adaptive learning(achieving the best published results to date - as far as we know - in a classic reinforcement learning task, the mountain car problem, see the results section). Second, we show that that an agent that has internalised a model of the environment can achieve better performance than an agent that simply encodes a policy, even if the model is imperfect, as long as it tries to reconcile the error between the internal model and the real world. Finally, we treat reactive control as merely a strong prior over reflective control, which might possibly allow one to gracefully increase the quality of an agent in an online fashion.

The rest of the paper is organised as follows: A background section introduces the world the agent is embodied in alongside the technical components that make up each agent. A methodology section where we describe how the aforementioned components are linked to create agents; A results section, where we present our findings, and finally a

[1]The term "forward model" is overloaded, see [14] for more details. In the particular experiment peformed in this paper, sensory and state inputs are the same, so we can avoid further clarification

discussion section where we propose further steps and reflect on the the whole paper.

II. Background

A. Optimisation and control

The problem of an agent finding an optimal policy in a certain environment is usually studied under control theory or biological action-selection. In case a generic solution to the problem is sought, it is generally attacked in the broader A.I/Alife community using mainly two different search methods. The first one is termed "reinforcement learning"[17] and it involves finding an intermediate function to the problem, called the value function, and using that function to find the optimal policy. More formally:

$$V(s) = E[R, s|\pi] \tag{1}$$

- s being the current state.
- $E[.]$, the expected value.
- R, the reward function.
- π a policy
- $V(.)$ the value function

The second approach, more popular among evolutionary robotics and alife practitioners, involves searching directly the policy space for some optimal policy π^*. In this context a process that resembles natural evolution[18] is used to shape the agent's control system. After a number of evolutionary generations, the resulting agents should develop a policy that can accommodate them in the world, hopefully in a successful manner.

Both approaches have been applied successfully to a wide range of problems. As a general remark, value function methods (ex. TD-learning, Q-Learning[19]) seem to find good solutions faster, while evolutionary methods converge to better solution on the long term[20]. It has been speculated that this is due an inability of the current reinforcement learning algorithms to make good use of generic function approximators.

B. The mountain car problem

We embody and measure our agents in a world formulated by the equations of the mountain car problem[21], which is commonly used as a benchmark/reference task in reinforcement learning. The problem involves an agent trying to reach the top of a hill from a random initial position and speed (the landscape can be seen in 1).

The state equations the govern the Mountain car world are as follows

$$\begin{aligned} v_{t+1} &= v_t + 0.001a_t - 0.0025cos(3x_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned} \tag{2}$$

The upper and lower bounds for the state variables are $-1.2 < x < 0.6$ and $-0.07 < v < 0.07$. Action is one of the $a \in [1.0, 0, -1.0]$, and represents acceleration Finally,

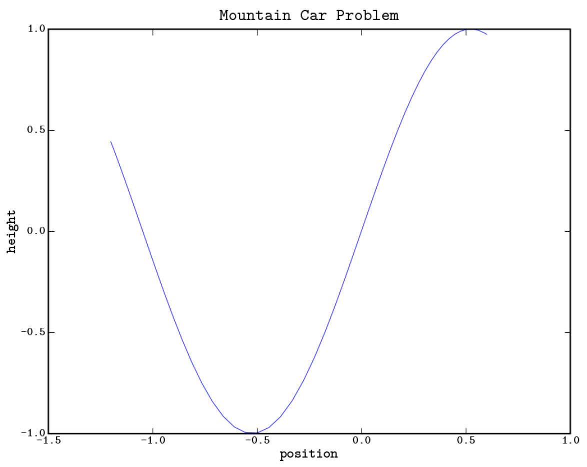

Fig. 1. Mountain Car Landscape

when x reaches a bound, v is set 0 (this has the effect of a car hitting a wall). The goal is to reach $x_{t+1} > 0.5$[2]

Depending on the initial position and speed, a straightforward approach of driving forwards does not work, as the motor of the car does not have the necessary force to directly overcome gravity. Thus a successful agent, depending on the initial position and speed, might need to move back and forth in the valley in order to get enough speed to beat gravity and get on top of the hill. A good policy will require a minimum number of steps, whereas a bad one might involve a big number of backwards-forwards moves.

C. Agent Components

All agents share a set of similar components and training methods, which we present below

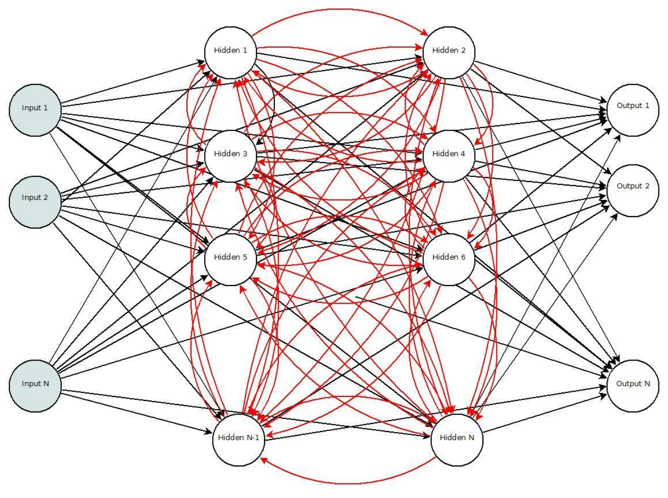

Fig. 2. Architecture of a sample CTRNN input nodes project to fully connected hidden nodes, which project to output nodes

[2]The version of the problem used in this paper is the one found here: http://www.cs.ualberta.ca/ sutton/MountainCar/MountainCar2.cp.

1) CTRNN: A Continuous Time Recurrent Neural Network (CTRNN) (see figure 2) is a neural network, which as it name implies, is recurrent and portrays a chaotic behaviour in time[22]. The way these networks work is by initialising the variables of the network to some random state. After some time the network will fall into its *attractor* position. The dynamics of the network are described in the following equation:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ji} \tanh(y_j - \theta_j) + I_i(t) \tag{3}$$

- $i = 1, 2, 3, \ldots, N$, with N being the number of nodes.
- y_i, the state of the node i, this can be thought of as the 'output' .
- w_{ji}, the strength of connection from j node to i node.
- The hyperbolic tangent $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$.
- $\tau_i > 0$ being the time constant of each node.
- θ_j being a bias term.

Euler approximation is used in order to perform the integration for each time step with $\delta t = 0.05$.

2) Evolution: Whenever evolution is mentioned (see the methodology section below) an $(\mu, \lambda) - CMA - ES$ [23] *Evolutionary Strategy* is used, where the parents μ for each generation are always half the number of the total population $\lambda = 10$. Our use of CMA-ES is mainly motived by its excellent performance in previous applications. In that respect, any Evolutionary algorithm or genetic algorithm that supports a real-valued encoding could be used. In our experiments, the evolutionary substrate is always a CTRNN. Thus, The genome is a real-valued and is a direct encoding of the CTRNN parameters. Each gene in the genome is initialise to 0.5, and $CMA - ES$ is configured to expect a standard deviation of 0.13 . When the genotype is transformed to a phenotype, its gene is assigned to either a weight, a bias or time constant τ. The process of decoding the gene binds each weight and bias to $[-10, 10]$ and τ to $[0, e^{4.0}]$. In this encoding, for a network if size N, we have $N^2 + 2N$ size genome.

3) Support Vector Machines: For this study we use the SVMs as a black box method for performing any necessary supervised learning. A more biologically plausible solution would be using liquid state machines[24], although the ubiquitousness and ease of use of SVMs makes them an easy choice. The implementation was done with libsvm[25], using the nu-svm SVM with the default libsvm RBF kernel [26]. The inputs were NOT scaled or pre-processed in any way.

III. Methodology

We have evolved three different agents, which all share a similar structure concerning their input and output signals. The fitness function used is the same for all the agents.

$$f = \sum_{s=1}^{s=100} t_{end} \tag{4}$$

where t_{end} is the number of timesteps it takes to reach the goal ($x > 0.5$). Thus the "better" agent is the one that reaches the goal faster. A fixed number of initial positions is generated randomly and uniformly from the state values ($-1.2 < x < 0.5$, $-0.07 < v < 0.07$). These sample are represented in the fitness function with s. Note here that the lower the fitness function gets, the better the adaptive value of an individual.

A. The agents

All agents sense their world in a perfect fashion (i.e. there is no sensory noise). All policy model parts of our agents are effectively CTRNNs. They encode two sensory channels; position (I_p^0) and velocity(I_v^1). They also control the power of the car engine by outputting a force (or acceleration) a, where

$$a = \begin{cases} 1, & \text{if } O_a^1 > 2/3 \\ 0, & \text{if } -2/3 \leq O_a^1 \leq 2/3 \\ -1, & \text{if } -2/3 < O_a^1 \end{cases} \tag{5}$$

where O_a^1 is the output of the $1st$ output neuron. The policy part of the agent is always a CTRNN. One must note here that the sensory information and the state are the same in the mountain car problem. A schematic of the agents can be seen in 3.

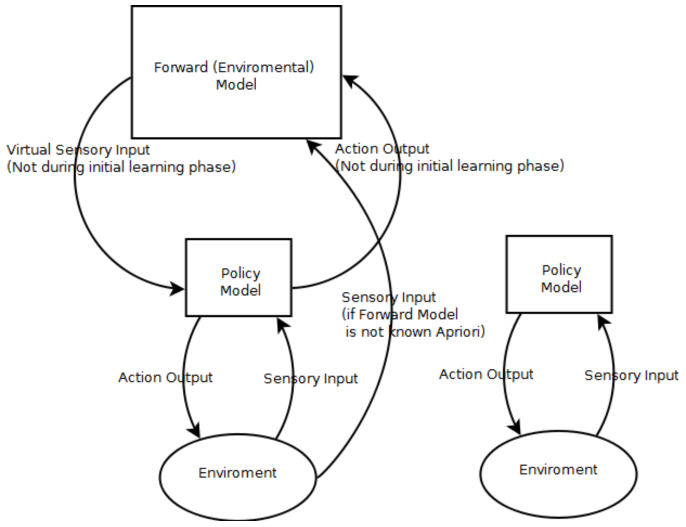
Forward (Enviromental) Model
Virtual Sensory Input (Not during initial learning phase)
Action Output (Not during initial learning phase)
Policy Model
Sensory Input (if Forward Model is not known Apriori)
Action Output
Sensory Input
Enviroment
Policy Model
Action Output
Sensory Input
Enviroment

Fig. 3. A forward model agent (left) and a policy only agent(right)

1) Policy Only agent (PO): This is the straightforward agent, where training consists of evolving an agent to try and reach the goal from various initial positions. This can be seen as the "classic" neuro-evolutionary agent, where a CTRNN is evolved for a number of generations against a fitness function (see above).

2) Perfect Forward Model agent (PFM): This is a fictional agent, where somehow the model of the world is known to the agent a priori. In this case, the agent still trains on the world as normal, but during normal operation/testing, the agent, in a set of actions vaguely reminiscent of neural

Darwinism[27] performs a short evolutionary run using the forward model. The equations of the forward model are identical to the ones that govern the dynamics of the state-space the agent was evolved in. Thus the agent "thinks" for a finite amount of time before making any move, simulating possible future rewards (using equations identical to the ones the environment uses in this case) and selecting the moves that do better. The evolution here is specialised and does not try to identify a generic agent, but rather tries to identify an agent that can efficiently solve the problem from a specific state. Effectively this means that the agent "overfits" an initial network that was discovered during the training evolutionary run to specific solution.

An alternative view of the setup above is to imagine that the policy of this agent is merely a very strong prior. It is the default behaviour of the agent provided no thinking time is present. The more time the agent has to improve on this prior, the better is should perform in any task.

3) Acquired Forward Model agent (AFM): This agent acquires an internal world model and then uses that model to predict some longer term reword. The architecture of the agent here is identical to the "Perfect Model Agent" presented previously, with the only difference being that the agent has acquired that model. During the training phase, and while the agent tries to identify a policy in the world, training examples are implicitly generated by the agent performing actions. These training examples are used to create a forward model of the world.

In order to learn the forward model, support vector machines(SVMs)[28] in regression mode are used for each sensory channel (position, velocity and reward). The input to the SVMs is x_t, v_t, a_t and the output is either x_{t+1}, v_{t+1} or r_{t+1}[3]. Due to the fact that the agent sees a very small number of positive rewards(r_t), we only add a negative reward to the examples for every one positive. The inputs were NOT scaled or pre-processed in any other way.

While testing, each action planned is executed both in the internal model and at the world. The error between the real world and the forward model is calculated as follows:

$$\begin{aligned} E^i_{abs} &= |(S^p_i - S^a_i)| \\ E^i_{max} &= |(S^{min}_i - S^{max}_i)| \\ E^i &= \frac{(E^i_{abs})}{(E^i_{max})} \\ \mathbf{E} &= [E_0, E_1, \ldots, E_n] \end{aligned} \tag{6}$$

This binds the error to $0 < E_i < 1.0$. In equation 6, E^i_{abs} is the absolute error for sensory input i, S^p_i is the current state for sensory input i, S^p_i is the projected/simulated state for input i , S^{min}_i is the minimum possible value for input i and S^{max}_i is the maximum value for input i. If the error becomes greater than some predefined prediction error E^b_i than the agent stops, recalculates the steps needed to reach the terminal position and proceeds as normal. Thus, the agent has a concept that its forward model is NOT perfect and tries to accommodate its observed reality, adjusting its behaviour if needed.

[3]Rewards are interpreted as an internal sensory channel.

TABLE I
RESULTS FOR 1000 RANDOM INITIAL POSITIONS

	PO	PFM	AFM
Number of test samples	1000	1000	1000
Mean	68.006	47.252	52.606
Min	1	1	1
Max	300	134	172

IV. RESULTS

We evolved all agents for 200 generations. The AFM agent was trained using 50000 examples generated during the original evolutionary training phase. We used a population of 10 individuals. In figure 4 one can see that the the landscape is quite "rugged", with the best genotype's fitness value "jumping" when improving. The process results in identical policy models for each type of agent (which the agents that use a forward model update online in fast 10-generations evolutionary run later on).

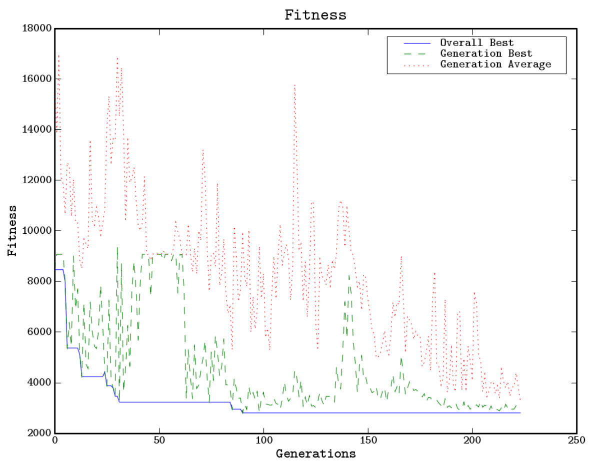

Fig. 4. Sample run results using $(5/5_w, 10) - CMA - ES$

During testing, we tested each agent for 1000 randomly generated initial positions. If an agent failed to achieve the goal within 2000 steps, the test case was declared a failure (though this never happened in practice). In table I PO stands for Policy Only, PFM for Perfect Policy Model and AFM for Acquired Policy Model. What is obvious from the table I is that an agent that encodes a forward model can achieve much better results, even though the forward model is necessarily imperfect. The PFM agent achieves the best results published to our knowledge for the problem so far(ex. [29], [30]). Note here that this approach required encoding a perfect forward model, something quite hard to acquire, unless the model is known in advance (ex. in most games).

Figures 6 and 5 contrast the difference between an agent with a perfect model and one the imperfect one. The agent

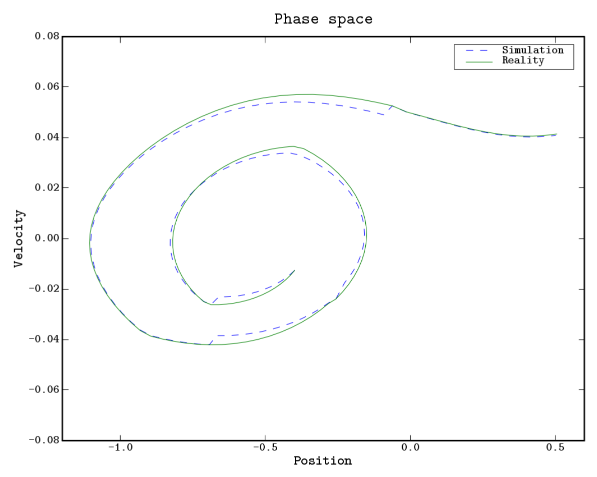

Fig. 5. Phase portrait of AFM with starting position $P = -0.398$,$v = -0.0127$. The imperfections in the forward model lead to correction (jumps) in the phase space of the simulation.

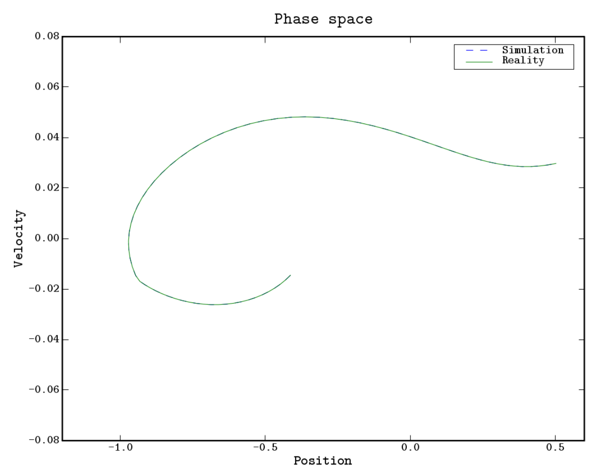

Fig. 6. Phase portrait of PFM agent with starting position $P = -0.398$,$v = -0.0127$. Simulation and reality are the same, so no error correction is needed.

with the acquired forward model "recalculates" its policy once one of the sensory channels error becomes higher than $E_s > 0.1$. The PO agent (not shown in any figure) achieves the same goal in 300 steps (its worst performance). This is probably due the fact that the policy agent encodes a simple policy of going back and forth until it reaches its goal, which is quite generic, but not optimal for every scenario.

V. Discussion

The goal of this paper has been to show that one can reason effectively using a forward model, even if the model if imperfect. The forward model seems to complement inefficiencies in the encodings of the neural network, boosting performance at the expense of computation time. Using a perfect forward model we have achieved the best published results on the mountain car problem. Our method can be used complementary to many different optimisation scenarios.

A case of interest would also be to formalise the situation where the agent does not feel certain about the world and ignores its internal model completely or partially, trying to acquire the strategy from the real world. This would lead to a more "natural" distinction between training and testing, as there would be no need to make training explicit, the agent would just act in the world as it sees fit, but after acquiring enough experience it would start reasoning using the internal model. A good question here is why we cannot find an optimal policy no matter how much training we do (which is evident in both reinforcement learning and evolutionary approaches to the mountain car problem in previous studies). Our proposal is that the current generation of machine learning cannot capture the underlying non-linearities in an optimal fashion, and this adversely affects the performance of any learning algorithm.

An alternative approach in order to achieve good performance has been the evolution of both the weights and the structure of the network. The problem with this approach is that it requires an increasingly more complex network in order to accommodate the complexity of the policy. We believe our approach of an initial somewhat correct but versatile behaviour that is updated online in a fast run might yield better results due to the fact that we actually try to approximate a large number of simple functions rather than a potentially complex one.

With this in mind, one can see adaptive/reactive control as merely a strong prior over anticipated/reflective behaviour. The longer-term goal of this research is to be applicable in solving computer games, where performance is a key requirement. In these scenarios, one has a perfect model of the world, but it seems that in more complex worlds reinforcement learning and GAs fail to evolve optimal solutions (rather one gets "good enough" results). What one would ideally want is optimal performance, which we think can be achieved by doing a very short evolutionary run on an already good approximate solutions. As long as we can keep the computational demands of the online run short, one might hope to optimise performance on the expense of some cpu power.

VI. Acknowledgements

This research is partly funded by a postgraduate studentship from the Engineering and Physical Sciences Research Council.

References

[1] J. Von Neumann, "Zur Theorie der Gesellschaftsspiele Math," *Annalen*, vol. 100, pp. 295–320, 1928.

[2] J. Togelius, S. Karakovskiy, J. Koutnık, and J. Schmidhuber, "Super Mario Evolution," in *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, 2009.

[3] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[4] K. Friston and K. Stephan, "Free-energy and the brain," *Synthese*, vol. 159, no. 3, pp. 417–458, 2007.

[5] K. J. Friston, J. Daunizeau, and S. J. Kiebel, "Reinforcement learning or active inference?" *PloS one*, vol. 4, no. 7, 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/19641614

[6] P. Verschure, T. Voegtlin, and R. Douglas, "Environmentally-mediated synergy between behaviour and perception in mobile robots," *Nature*, vol. 425, pp. 620–624, 2003.

[7] R. Conant and W. Ashby, "Every good regulator of a system must be a model of that system," *International journal of systems science*, vol. 1, no. 2, pp. 89–97, 1970.

[8] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science: A Multidisciplinary Journal*, vol. 16, no. 3, pp. 307–354, 1992.

[9] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the Seventh International Conference on Machine Learning*, vol. 216, 1990, p. 224.

[10] D. Silver, R. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 968–975.

[11] R. Dawkins, "The evolved imagination: Animals as models of their world," *Natural History magazine*, vol. 104, pp. 8–11, 1995.

[12] J. Tani and S. Nolfi, "Learning to Perceive the World as Articulated: An Approach for Hierarchical Learning in Sensory-Motor Systems, 5th Int," in *Conf. on Simulation of Adaptive Behavior (SAB-5)*, 1998, pp. 270–279.

[13] G. Baldassarre, "Forward and bidirectional planning based on reinforcement learning and neural networks in a simulated robot," *Anticipatory Behavior in Adaptive Learning Systems*, pp. 87–106, 2003.

[14] A. Karniel, "Three creatures named forward model," *Neural Networks*, vol. 15, no. 3, pp. 305–307, 2002.

[15] D. Wolpert, Z. Ghahramani, and M. Jordan, "An internal model for sensorimotor integration," *Science*, vol. 269, no. 5232, p. 1880, 1995.

[16] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[17] R. Sutton and A. Barto, "Reinforcement learning," *Journal of Cognitive Neuroscience*, vol. 11, no. 1, pp. 126–134, 1999.

[18] D. Goldberg and J. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.

[19] C. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[20] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber, "Ontogenetic and phylogenetic reinforcement learning," *Zeitschrift Künstliche Intelligenz*, 2009.

[21] R. Button and A. Barto, "Reinforcement learning: an introduction," 1998.

[22] R. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, p. 469, 1995.

[23] N. Hansen, "The CMA evolution strategy: a comparing review," *Studies in Fuzziness and Soft Computing*, vol. 192, p. 75, 2006.

[24] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[25] C. Chang and C. Lin, "LIBSVM: a library for support vector machines," 2001.

[26] B. Scholkopf, A. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[27] G. Edelman, *Neural Darwinism: The theory of neuronal group selection*. Basic Books New York, 1987.

[28] M. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent systems*, vol. 13, no. 4, pp. 18–28, 1998.

[29] S. Lucas, "Temporal Difference Learning with Interpolated Table Value Functions," in *IEEE Computational Intelligence and Games*, 2009.

[30] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *The Journal of Machine Learning Research*, vol. 7, p. 917, 2006.