Forward and Bidirectional Planning Based on Reinforcement Learning and Neural Networks in a Simulated Robot

Gianluca Baldassarre

Computer Science Department, University of Essex,
CO4 3SQ, Colchester, United Kingdom
Institute of Cognitive Sciences and Technologies,
National Research Council of Italy (ISTC-CNR)
Viale Marx 15, 00137, Rome, Italy
baldassarre@ip.rm.cnr.it

Abstract. Building intelligent systems that are capable of learning, acting reactively and planning actions before their execution is a major goal of artificial intelligence. This paper presents two reactive and planning systems that contain important novelties with respect to previous neural-network planners and reinforcement-learning based planners: (a) the introduction of a new component ("matcher") allows both planners to execute genuine taskable planning (while previous reinforcement-learning based models have used planning only for speeding up learning); (b) the planners show for the first time that trained neural-network models of the world can generate long prediction chains that have an interesting robustness with regards to noise; (c) two novel algorithms that generate chains of predictions in order to plan, and control the flows of information between the systems' different neural components, are presented; (d) one of the planners uses backward "predictions" to exploit the knowledge of the pursued goal; (e) the two systems presented nicely integrate reactive behavior and planning on the basis of a measure of "confidence" in action. The soundness and potentialities of the two reactive and planning systems are tested and compared with a simulated robot engaged in a stochastic path-finding task. The paper also presents an extensive literature review on the relevant issues.

1 Introduction

Since its birth, from the first experiments with the robot Shakey [8] to current research in planning [1] and behavior based robotics [6] [2], a major goal of artificial intelligence has been building intelligent systems that are both capable of acting in a reactive fashion and planning the course of action before its execution. In fact reactive behavior allows agents to quickly react to dynamic and unpredictable events. In order to be "tuned" with reality, the reactive behavior has to incorporate forms of "implicit anticipation" (cf. Butz et al. in this volume). On the other hand, the capacity to predict future states of the world on the basis of its regularities allows agents to be flexible

and taskable, i.e. to re-use knowledge to pursue different goals (see below, and see the concept of "state anticipation" of Butz et al. in this volume).

Traditionally, artificial intelligence planning systems have been based on logical information representations built a-priori by the designer [1]. When these planning systems are used to control robots, sensors' readings are converted into logical representations, the control is implemented in terms of manipulations of these representations, and the outcome of this processing is again converted into effectors' commands. This approach has difficulties as the time-consumption of logical reasoning about the effects of low-level actions is too expensive to generate real-time behavior [21].

At least initially, behavior-based robotics proposed to eliminate logical deliberation and planning from control altogether [6]. Reactive behavior was implemented through numerical functions and/or rules that "directly" linked the sensors' numerical patterns to the effectors' numerical patterns. More recently, behavior-based robotics has attempted to integrate reactive behavior based on numerical representations and planning (usually) based on logical representations (see review in [2]). However, the resulting systems have an important limitation: they imply a *double recoding* of information, from a numerical format to a logical format and vice versa, that is difficult to implement, slow and prone to errors.

The motivation of this research was building reactive and planning systems that rely only on numerical representations. This novel approach should allow coping with noisy and unpredictable environments through reactive behaviors, having the flexibility of planning, and at the same time avoiding the problem of the interface between different information representation formats. Given the level of development that they have reached (cf. 9), neural networks have been chosen for this purpose. The attempt of this research is challenging and interesting at the same time. In fact its solution implies answering a number of questions of the following type: What kind of information representations can be used to plan with neural networks? What is the origin of this information (hand-coded, experience)? How implementing the loop "decision of action – prediction of consequences – decision of action - …" required by planning with neural networks? How can the neural planning process be used to influence future action? What are the advantages and disadvantages of using neural networks versus logic-based algorithms to implement planning?

The systems presented here are based on dynamic programming and reinforcement learning methods. Roughly speaking, dynamic programming [20] is based on a model of the world that indicates which states and rewards, e.g. rewards associated to goal states and costs associated to other states, follow the execution of one of the available actions. Dynamic programming repeatedly and systematically explores all the states of the model of the world to associate an "evaluation" (e.g. a number between 0 and 1) with each of them on the basis of the rewards and the states' "contiguity" in time. This evaluations, that are gradually higher (or lower) for states closer to the goal states, form a gradient field. Actions are selected so as to ascended (or descend) this gradient field. The way this is done is called "action-selection policy". As dynamic programming, to which they are closely related, reinforcement learning methods build a gradient field of evaluations over the states of the world [25]. However, differently from dynamic programming, they usually do not use a model of the world ("model-free reinforcement learning methods"). Instead, they learn the evaluations by directly exe-

cuting the actions in the world and by observing the consequences in terms of new states and rewards (costs) experienced. Dynamic programming and reinforcement learning methods are probably the most suitable framework currently available to implement planning with neural networks. In fact, even if developed for whole state representations, many interesting applications in these fields "break" the representation of states into feature patterns. Feature patterns, being distributed and parallel information representations, can be readily processed by neural networks. Moreover, both approaches are based on state evaluations that can be easily represented with neural networks (see [3] and the systems presented here).

Sutton [26] has integrated planning based on dynamic programming and reinforcement learning into a class of architectures called "Dyna" (from "dynamic programming"). The central idea of Dyna architectures is to use a model of the world to generate "simulated experience" for reinforcement-learning training (e.g. [14]). The algorithms proposed here are inspired by Dyna-PI architectures, a class of Dyna architectures based on actor-critic reinforcement-learning methods [25] ("PI" stands for "policy iteration", the process at the basis of actor-critic methods). Actor-critic methods, differently from other reinforcement-learning methods that select actions on the basis of the evaluations, use a data structure that stores the action-selection policy in the form of probabilities of selecting the actions in correspondence to different states.

To the best of the author's knowledge, so far Dyna architectures have been only used to speed up planning (e.g. [14], [19] and [26]) and not to implement genuine "taskable planning". Let us define a goal as a state to reach. A system is capable of executing taskable planning if it possesses a model of the world and can use the goal-independent information stored in it to autonomously reach a goal, never reached previously, more efficiently than a purely reactive system (see below). Dyna architectures are not taskable because they need a model of the environment that incorporates the rewards (or costs) associated with the goals [26]. As a consequence, when Dyna architectures are assigned a goal, they either need to learn the evaluations associated with that goal (but to do this they need to experience the goal several times, so they cannot be efficient since the first time they reach the goal) or the evaluations have to be hardwired by the designer into the model of the world (but in this case the systems are not autonomous). The systems proposed here are endowed with a computationally simple but theoretically important new component in order to overcome this problem: the "matcher". This is a simple neural network that compares the goal assigned to the system with the current state and *internally and autonomously* generates a reward if they are similar (cf. [4] for an hypothesis on the possible brain structures that might correspond to the matcher). Once capable of producing rewards internally, the systems can engage in planning by generating "simulated experience" to train their reactive components. Simulated experience is generated by producing several "chains of prediction". To generate a chain of predictions the systems select an action in the current state, predict the state that would follow that action's execution, suspend the execution of this action, select another action starting from the predicted state, and so on. After doing this, when the systems act again in the world they can reach the goal with high efficiency compared to a system made up by the reactive components only.

There is an interesting issue related to the use of neural networks for planning. The predictions generated by the neural networks of the model of the world, are affected

by noise. As a consequence one would expect to observe an accumulation of noise if chains of predictions are generated. Surprisingly, we will see that there are some mechanisms for which this noise is filtered out so that long chains can be generated that maintain a correspondence with the images of the world.

While generating the chains of predictions through the model of the world, the systems learn by reinforcement learning *as if they were acting in the world*, and then they use the knowledge acquired in this way to act. Two novel algorithms are proposed to control these processes. These algorithms suitably interleave acting and planning according to the systems' "confidence" in action, defined on the basis of the action selection probabilities.

In a given moment, the systems know not only the current state, but also the goal state. The second planner proposed (bidirectional planner) alternately generates forward chains from the current state and backward chains from the goal. The bidirectional planner is more efficient than the forward planner because it is quicker in learning the evaluations and is more focused on the states relevant for the task while exploring the world's model.

Section 2 presents the reinforcement learning framework, the scenario and the simulated robot used to test the algorithms. Section 3 presents the forward planner and the results of the simulations run with it. Section 4 presents the bidirectional planner and compares its performance with the performance of the forward planner. Section 5 presents an extensive literature review and highlights the novelties of the algorithms presented here. Finally, section 6 presents the conclusions and the future work.

2 Reinforcement Learning Framework, Scenario of Simulations and Simulated Robot

The reinforcement learning framework, based on "Markov Decision Problems", is sketched here (see [3] and [25] for details). A Markov decision problem implies that an agent interacts with the world at discrete time steps t = 1, 2, 3, … On each time step t the agent perceives a state of the world $s_t \in S$, and on the basis of this selects an action $a_t \in A$. In response to each action the world produces a reward r_{t+1} and a new state with probability: $p^a_{ss'} = Pr[s_{t+1}.= s' \mid s_t.= s, a_t = a]$ for all $s, s' \in S$ and $a \in A$. The planners introduced below learn a simplified model of the world M: $(s_t, a_t) \rightarrow s_{t+1}$. This model is deterministic and does not take into considerations the rewards that are universally modeled by the matcher on the basis of the goal and the current state.

An (action selection) policy π is defined as a mapping from the states to the action selection probabilities, π: $S \times A \rightarrow [0, 1]$. For each state s a "state-evaluation function" $V^\pi[s]$ is defined that depends on the policy π and is calculated as the expected discounted future reward from s:

$$V^\pi[s] = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots] = \Sigma_{a \in A}[\pi[a, s] \Sigma_{s' \in S}[p^a_{ss'} (r_{t+1} + \gamma V^\pi[s'])]] \quad (1)$$

where $\pi[s, a]$ is the probability that the policy selects a in s, E[.] is the mean operator, and $\gamma \in [0, 1]$ is a discount coefficient. The agent aims at finding an "optimal policy" that maximizes $V^\pi[s]$ for all $s \in S$.

The algorithms presented here apply to a restricted class of Markov decision problems, the "stochastic shortest-path problems" (cf. [5]). In these problems the reward is equal to 1 for a *unique* "goal state" s^g and equal to 0 for all other states. The agent aims at finding a policy that leads from a start state s^s to the goal state s^g with the minimum number of steps. This restriction is introduced to allow the systems presented here to generate an internal reward in correspondence to the goal through the matcher. This simplification does not reduce the generality of the systems: more complex situations might be dealt with by using more complex internal reward generators (e.g. a component that generates costs when actions are selected).

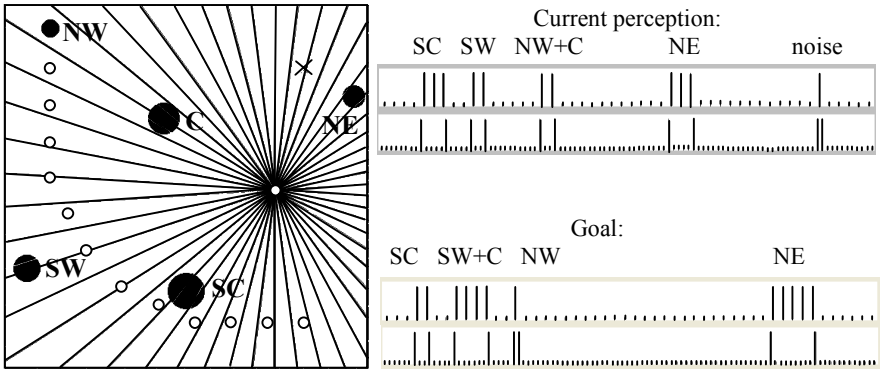

Fig. 1. *Left*: The scenario of the simulations containing five landmarks (*black circles*), the scope of the simulated robot's 50 visual sensors (*delimited by the rays*), the simulated robot (*circle at origin of rays*), the 12 start positions (*white circles*), and the goal (*marked with an* X) used in the tests. Letters indicate different landmarks. *Top right:* The sensory pattern generated by the simulated robot's sensors (at the position shown in the graph on the left) and the corresponding contrast pattern. Letters indicate the landmarks that affect the different portions of the image. *Bottom right:* The sensory pattern and contrast pattern corresponding to the goal

Now the scenario and the simulated robot used to test the algorithms are described. The scenario (Fig. 1) is a square arena with sides measuring 1 unit and with 5 circular landmarks inside. These landmarks are also obstacles for the simulated robot. The simulated robot can see the landmarks with a one-dimension horizontal retina covering 360 degrees. Notice that the simulated robot cannot perceive distances, cannot see a landmark that is behind another landmark, and perceives just a "big" landmark if there are two or more landmarks that are contiguous in sight. This implies that different states might generate the same image. Problems of with these features, called "partially observable Markov decision problems", might raise difficulties for reinforcement learning that are not considered here (see [10] and [13] on this).

Markov processes are based on whole state representations. However here, as in the most interesting applications of reinforcement learning, the simulated robot perceives the world through sensors that return a feature-like pattern (vector) $\mathbf{x}$ in correspondence to the state s (this also implies that s^s and s^g become $\mathbf{x}^g$ and $\mathbf{x}^s$). More in details, the simulated robot has a retina made up by 50 units $\mathbf{x}$. Each unit x_i activates with 1 if a landmark is in its scope, with 0 otherwise. This activation is affected by noise (0.01

probability of flipping for each sensor). The signals coming from the retina are always aligned with the magnetic north through a simulated compass before being fed to the systems. This facilitates the task by making the images rotation invariant. The reading of the compass is affected by Gaussian noise (0 mean, 1 degree variance). Before being sent to the controllers, the retina signals are re-mapped into a vector **y** of 100 binary units representing the image "contrasts" (contrasts between the landmarks, perceived as "black", and the "background", perceived as "white"). Two contiguous retinal units give unit activation to one contrast unit y_j if they are respectively on and off, to another contrast unit if they are respectively off and on, and to no contrast units if they are both on or both off. This simple re-mapping implements an expansion of the input space that allows the systems to work properly by using simple two-layer networks for the controller [3] (more complex tasks would require a more powerful preprocessing). At each cycle of the simulation the system has to select one of eight actions, each consisting of a 0.05 step in one of eight directions aligned with magnetic north (north, northeast, etc.). The outcome of these actions is affected by Gaussian noise with 0 mean and 0.01 variance. If the simulated robot moves against the arena's boundaries or the obstacles it "bounces back", i.e. it is set at the previous position.

3 Reinforcement Learning and Forward Planning

Fig. 2 shows the reinforcement-learning (reactive) and the planning components of the bidirectional planner. The forward planner can be obtained from the bidirectional planner by not considering the components necessary to generate the backward prediction chains (these components have a dotted border in the figure). The forward planner is made up by the reinforcement learning components (bold border in the figure) and by the components used to generate the forward prediction chains (thin border in the figure). Now the components of the forward planner are analyzed in detail.

3.1 Actor Critic Reinforcement Learning

The reactive reinforcement learning components of the system are the critic (composed of the TD-critic and the evaluator), the actor, and the stochastic selector. In the introduction it was mentioned that the matcher is used to generate an internal reward signal while the system is planning. However, in order to facilitate comparisons, it will also be used to generate this signal when the planner will be used as a share reactive system in order to show that it does implement genuine taskable planning. For this reason the matcher is analyzed here together with the reactive components. The matcher is a hand designed neural network with 200 input units and 1 output unit. It returns 1 as output when the first part of its input (100 units encoding the goal, i.e. the contrast image of the landmarks from the goal position) has at least 94% of units with the same activation of the corresponding units of the second part (100 units encoding the current input contrast pattern, or the current prediction). Otherwise it yields 0 (cf. [3] for the details).

The "actor" is a two-layer feed-forward neural network that is fed with the input pattern $\mathbf{y}_t$ and has eight sigmoid output units that locally encode the actions. To select an action, the activation m_l (the "action merit") of the output units is sent to a selector that implements a stochastic winner-take-all competition. The probability Pr[.] that a given action a_g becomes the winning action a_w is given by:

$$\Pr[a_g = a_w] = m_g \; / \sum_l m_l \qquad (2)$$

In the scenario studied here the actors' sigmoid output units plus this selection method yield a behavior more robust than the popular soft-max function (cf. [3] and [25]).

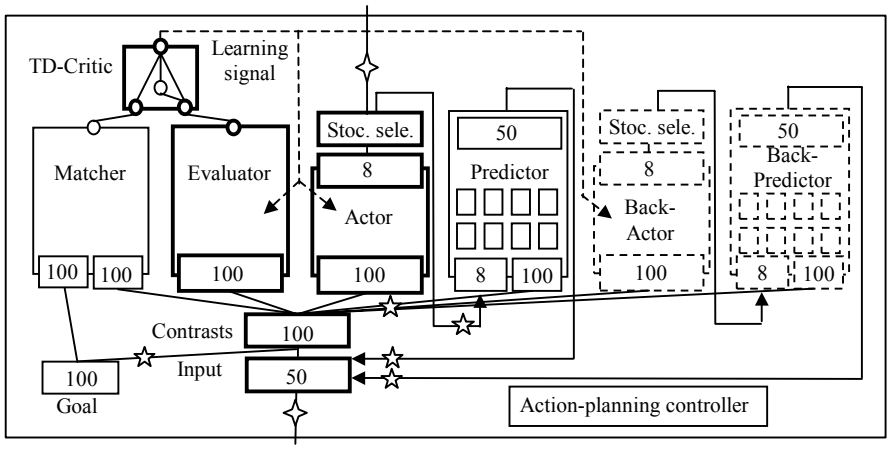

Fig. 2. The systems' architecture. Networks with a *bold*, *thin* and *dashed border* implement reinforcement learning, forward planning, and bidirectional planning respectively. *Arcs* and *arrows* respectively indicate forward and backward connections that "copy" a pattern from one layer to another. The *four and five spike stars* indicate the channels respectively set open and closed by the action-planning controller when acting (vice versa when planning). *Dashed arrays* indicate the learning signal used to update the weights of the evaluator, actor and back-actor.

The "evaluator" is a two-layer feed-forward neural network that is fed with the input pattern $\mathbf{y}_t$ and that returns the estimation $V'^{\pi}[\mathbf{y}_t]$ of the evaluation $V^{\pi}[\mathbf{y}_t]$ with its linear output unit. Similarly to what has been done for the states s, $V^{\pi}[\mathbf{y}_t]$ is defined as the expected discounted sum of future reinforcements r, given the current actor's policy π (cf. equation 1; γ is set at 0.95 in the simulations).

The "TD-Critic" is a neural implementation (see [3] and [4] for details) of the computation of the Temporal-Difference error e defined as (see [25]):

$$e_t = (r_{t+1} + \gamma V'^{\pi}[\mathbf{y}_{t+1}]) - V'^{\pi}[\mathbf{y}_t] \qquad (3)$$

Notice that $V^{\pi}[\mathbf{y}_t]$ is set at 0 if $\mathbf{y}_t = \mathbf{y}^g$ because a "trial" ends [3, 25]

The evaluator is trained with a Widrow-Hoff algorithm [31] that uses the error signal e_t. In particular, its weights w_j are updated as follows:

$$\Delta w_j = \eta \; e_t \; y_j = \eta \; ((r_{t+1} + \gamma V'^{\pi}[\mathbf{y}_{t+1}]) - V'^{\pi}[\mathbf{y}_t]) \; y_j \qquad (4)$$

where η is a learning rate (set at 0.1 in the simulations). This rule implies that the evaluator's estimation $V'^{\pi}[\mathbf{y}_t]$ is made closer to the target value $(r_{t+1} + \gamma V'^{\pi}[\mathbf{y}_{t+1}])$. This target is a more precise evaluation of $\mathbf{y}_t$ because it is expressed at time t+1 on the basis of the observed r_{t+1} and the estimation $V'^{\pi}[\mathbf{y}_{t+1}]$, usually closer to the goal than $V'^{\pi}[\mathbf{y}_t]$.

The actor is also trained with a Widrow-Hoff algorithm according to the TD-Critic's error signal. The updating of the merit values is done by adjusting the weights of the neural unit corresponding to the winning action a_w (and only this) as follows:

$$\Delta w_{wj} = \zeta\, e_t\, (4\, m_j\, (1 - m_j))\, y_j = \zeta\, ((r_{t+1} + \gamma\, V'^{\pi}[\mathbf{y}_{t+1}]) - V'^{\pi}[\mathbf{y}_t])\, (4\, m_j\, (1 - m_j))\, y_j \qquad (\mathbf{5})$$

where ζ is a learning rate (0.1 in the simulations) and $(4\, m_j\, (1 - m_j))$ is the derivative of the sigmoid function multiplied by 4 to homogenize the actor's and critic's learning rates. When a new goal is assigned to the simulated robot, the weights of the evaluator and actor are randomized in [-0.001, +0.001], so the evaluations expressed by the evaluator's linear output unit are around 0, and the merits (probabilities) expressed by the actor (stochastic selector) are all around 0.5 (0.125). This implies that initially the simulated robot randomly explores the world (or the model of the world). Afterwards, the evaluator shapes the evaluations on the basis of the rewards, and at the same time the actor shapes the action probabilities on the basis of the evaluations. The parallel training of the evaluator and actor is called "policy iteration" [25] and gives the name to the Dyna-PI architectures (cf. [26] and see below).

3.2 Forward Planning

This section illustrates the components added to the reinforcement learning components to have forward planning. The "predictor", i.e. the system's model of the world, is a set of 8 feed-forward two-layer networks, called "experts". Each expert corresponds to one action. Each expert takes $\mathbf{y}_t$ as input, and is specialized to predict the following sensors' activation $\mathbf{x}_{t+1}$, if the action corresponding to it is executed, with its sigmoid output units. To have a binary output, the output of each sigmoid unit is squashed to 0 if below 0.5, and to 1 if above. A hand-designed selector chooses the expert corresponding to the selected action to yield the output of the predictor. The experts are trained while the robot navigates randomly in the environment for 200,000 cycles. This brings the "quadratic error" (computed as $(\sum_i[(x'_i - x_i)^2]/n))^{1/2}$ where x_i and x'_i are the actual and predicted activations of the n sensors) to about 0.24. This training is done before the simulations illustrated below. At each cycle, the contrast pattern $\mathbf{y}_t$ and the input pattern $\mathbf{x}_{t+1}$, observed after the execution of one action, are respectively used as input and as teaching output to train the expert that selected the action with a Widow-Hoff rule [31]. Notice that the predictor yields deterministic predictions that tend to be the average of the $\mathbf{x}_{t+1}$ observed. This is a simplification since the world is stochastic and the model should produce stochastic predictions.

The action-planning controller is a hand-designed algorithm (Fig. 3) that decides the planning or acting mode of the system and generates the forward (and backward) chains. Fig. 4 shows the details of one cycle of action and one cycle of (forward or backward) planning. The whole algorithm of Fig. 3 and Fig. 4 is executed sequentially *one time at each cycle of the simulation*.

```
//Initialisations
01 IF(NewGoalHasBeenAssigned)
02   MaxStepsPlan := 1
03   ConfThresh := MaxConfThresh
04   ForwardPlanning := TRUE
05   StepPlan := 0
06   InputFromWorld := TRUE
//Decision about planning or acting
07 IF(InputFromWorld)
08   System gets input x_t (y_t) from the robot's sensors
09   Actor gets y_t and gives m_t
10   Confidence is computed on the basis of m_t
11   IF(Confidence < ConfThresh)
12     Planning := TRUE
13   ELSE
14     Planning := FALSE
15     ConfThresh := MIN(MaxConfThresh, ConfThresh + Gain)
//Control of forward chains' length and interruption
16 IF(Planning)
17   StepPlan := StepPlan + 1
18   ConfThresh := ConfThresh - Decay
19   IF(ForwardPlanning)
20     IF(InputFromWorld = TRUE)
21       InputFromWorld := FALSE
22     ELSE
23       System uses predictor's output y_t as input
24     IF(GoalReached OR StepPlan = MaxStepsPlan)
25       IF(StepPlan = MaxStepsPlan)
26         MaxStepsPlan := MaxStepsPlan + 1
27       ELSE
28         MaxStepsPlan := MIN(MaxStepsPlan, StepPlan * 2)
29       InputFromWorld := TRUE
30       IF(BidirectionalPlanning)
31         ForwardPlanning := FALSE
32         ForwardSteps := StepPlan
33         GoalAsInput := TRUE
34         InputFromWorld := FALSE
35       StepPlan := 0
//Control of backward chains' length and interruption
36   ELSE
37     IF(GoalAsInput = TRUE)
38       System uses goal y^g as input
39       GoalAsInput := FALSE
40     ELSE
41       System uses back-predictor's output x_t (y_t) as input
42     IF(StepPlan = ForwardSteps)
43       ForwardPlanning := TRUE
44       InputFromWorld := TRUE
45       StepPlan := 0
```

Fig. 3. Pseudo-code of the action-planning controller. ":=" is the assignment operator, "MIN(. , . , …)" is a function that returns the minimum argument, "//" indicates a comment

If the variable "BidirectionalPlanning" is set at "FALSE", then the algorithm implements the forward planner. Let us see how the forward planner works. When a new goal is assigned to the system, some variable settings are done (line 1 to 6). The sys-

tem's planning or acting mode (variable "Planning") is decided each time the system receives an input from the world (line 7) on the basis of the system's "confidence" (line 8 to 14).

```
//Networks' activation in one forward chain cycle
46 IF(Planning)
47   IF(ForwardPlanning)
48     Evaluator gets y_t and gives V'^π[y_t]
49     Actor gets y_t and gives m_t
50     Stochastic selector gets m_t and gives a_t
51     Predictor gets y_t, a_t and gives x_t+1 (y_t+1)
52     Matcher gets y_g, y_t and gives r_t
53     TD-Critic gets V'^π[y_t-1], V'^π[y_t], r_t, gives e_t-1
54     Evaluator gets y_t-1, e_t-1 and learns
55     Actor gets y_t-1, m_t-1, a_t-1, e_t-1 and learns
56     IF(BidirectionalPlanning)
57       Back-Actor gets y_t and gives m_t-1
58       Back-Actor gets y_t, m_t-1, a_t-1(actor), e_t-1 and learns
//Networks' activation in one backward chain cycle
59   ELSE
60     Back-actor gets y_t and gives m_t-1
61     Back-stochastic selector gets m_t-1 and gives a_t-1
62     Back-predictor gets y_t, a_t-1 and gives x_t-1 (y_t-1)
63     Evaluator gets y_t-1 and gives V'^π[y_t-1]
64     Matcher gets y_g, y_t and gives r_t
65     TD-Critic gets V'^π[y_t-1], V'^π[y_t], r_t and gives e_t-1
66     Evaluator gets y_t-1, e_t-1 and learns
67     Back-actor gets y_t, a_t-1, e_t-1 and learns
68     Actor gets y_t-1 and gives m_t-1
69     Actor gets y_t-1, m_t-1, a_t-1 (back-actor), e_t-1 and learns
//Networks' activation in one action cycle
70 ELSE
71   Evaluator gets y_t and gives V'^π[y_t]
72   Actor gets y_t and gives m_t (already done in line 9)
73   Stochastic selector gets m_t and gives a_t
74   Matcher gets y_g, y_t and gives r_t
75   TD-Critic gets V'^π[y_t-1], V'^π[y_t], r_t and gives e_t-1
76   Evaluator gets y_t-1, e_t-1 and learns
77   Actor gets y_t-1, m_t-1, a_t-1, e_t-1 and learns
78   System executes a_t in the world
79   IF(BidirectionalPlanning)
80     Back-Actor gets y_t and gives m_t-1
81     Back-Actor gets y_t, m_t-1, a_t-1 (actor), e_t-1 and learns
```

Fig. 4. Pseudo-code of the activation of the neural networks components in one cycle of forward planning, backward planning and action

The system's confidence is defined as the highest of the actions' probabilities measured at the position currently occupied by the simulated robot. If the confidence is above a certain threshold the system acts in the world and the predictor is not used (cf. Fig. 4). If the confidence is below the threshold, the action-planning controller "disconnects" the robot from the world (line 11, 12, 16, 22 and 23; see 4 spike stars in

Fig. 2), in the sense that it starts to generate simulated experience by using the predictor and the matcher (cf. Fig. 4).

Each chain of predictions starts when the system switches from the acting to the planning mode (line 7, 8, 11, 12), and starts from the image that corresponds to the position currently occupied by the simulated robot (line 7 and 8). Then the chain continues with "rings" each made up by a prediction from the predictor (one for each cycle: line 19 to 23). Notice that chains of predictions tend to be different between them since the system selects actions stochastically (Fig. 4, line 49 and 50,). If one chain terminates without encountering the goal, the succeeding chain gets one "ring" longer (line 2, 24 to 26), otherwise it tends to get shorter (line 25, 27 and 28). While planning, the confidence threshold decreases (line 18). This prevents the robot from getting stuck in places in which the system is unable of becoming "confident" enough to start to move (for example, without this mechanism the simulated robot got stuck between the arena's border and the northwest landmark). While acting, the threshold increases again and reaches the maximum level without ever exceeding it (line 15). This guarantees that the simulated robot tends to move only when the confidence is above the maximum level of the threshold (MaxConfThresh). In the simulations the parameters that regulate these aspects are set as follows: Decay = 0.000001, Gain = 0.01, MaxConfThresh = 0.15. Each time a chain of predictions is terminated (either because the goal has been encountered or because it has reached a maximum length, line 24) the system "connects" again to the sensors (line 24, 29, 7 and 8), updates the mode (line 9 to 14), and starts to act or to generate another chain of predictions.

Fig. 4 shows the activation of the neural networks executed in one cycle of action when the variable "planning" is "FALSE" (line 46, 70 to 78). When planning is "TRUE" the predictor produces one of the predictions that make up the chain of predictions (line 51) and the matcher checks if the chain encounters the goal (line 52). Moreover, while the chains of predictions are generated the actor and the evaluator are trained with reinforcement learning as if the robot were acting in the world (compare line 48 to 55 with line 71 to 77). This allows the evaluator to improve its evaluating capacity and the actor to shape the action probabilities. As a consequence, when the system stops planning and acts in the world (line 78) it reaches the goal following a path that tends to be straight.

3.2 Reactive Behavior and Forward Planning: Results and Interpretations

The first simulation has tested the taskability of the reactive-planning system. This has been done by comparing its performance with the performance of the system made up by the reactive components only. The simulated robot is set at the southeast start, and its task is to reach the goal. Each time the simulated robot reaches the goal it is set at another *randomly-drawn* position of the arena. This is done for 50,000 cycles.

Fig. 5 reports the results of this simulation (averaged over 10 simulations run with different random seeds). For both the reactive and planning controllers the number of moves taken to reach the goal was measured and plotted against the cumulated cycles (this measure was sampled every 100 cycles, and then smoothed with a 10-step moving average). Each cycle reported in the graph implied the execution of one action and

eventually, if the controller was planning, a variable number of planning cycles. In the case of the planner, the graph also reports the number of planning cycles per move.

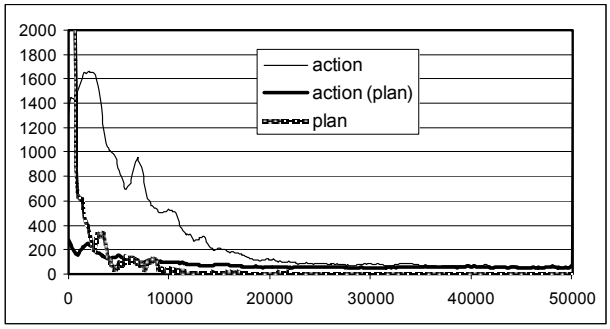

Fig. 5. *Y-axis*: moves per success taken by the reactive system (*action*), and moves per success (*action (plan)*) and planning cycles per success (*plan*) taken by the reactive-planning system, measured against the cumulated simulation cycles (*x-axis*). The measures have been sampled every 100 cycles and have been smoothed with a 10-step moving average, and are the average of the results of 10 simulations run with different random seeds

When assigned the (new) goal, the reactive system reaches it, by random walk, in about 1600 steps. When assigned the (new) goal, the reactive-planning system reaches it in about 200 steps *from the first time it pursues it.* This result is achieved through a considerable amount of planning processing: on average (10 random seeds) it takes 40,116 planning cycles to reach the goal the first time. During this planning activity the skills of the evaluator and actor improve so that when the system decides to act in the world it can achieve the goal by following an almost straight line. If the confidence threshold is set at 0.25 (a value higher than the previous 0.15), the performance of the planner improves to about 50 moves (see Fig. 6). These results demonstrate that the reactive-planning system implements genuine taskable planning. In fact, by using the goal-independent information stored in the predictor, the planner outperforms the corresponding reactive system from the first time it pursues the goal.

Fig. 5 shows another important property of the reactive-planning system. With repeated trials the system needs progressively less planning cycles, until it is capable of reaching the goal reactively without planning in about 40 moves from any position of the arena (the optimal path, not considering noise and obstacles, is about 15 moves). Interestingly, direct observation of the simulated robot's behavior also showed that once the robot has planned from a given position and then starts to act, it stops acting only if it reaches regions that are far away from the path between that position and the goal. This happens because planning process is focused on the states between the position where planning takes place and the goal. These results indicate that the algorithm nicely interleaves action and planning on the basis of "confidence", and that while planning it develops the skills of the evaluator and actor only for relevant states.

The quality of the chains of predictions generated by the predictor is a critical aspect of the system. Recall that when the predictor was trained, the square error per output unit did not go below 0.24, a quite high level if one considers that the error made by drawing a prediction randomly is about 0.5. On the basis of this, one could

reasonably expect that the chains of predictions would have become completely random after few steps. In fact if a noisy prediction is fed back into the predictor, one would expect that the new prediction accumulates a double amount of noise, and so on for the further predictions generated along the chain. Interestingly, this is not the case.

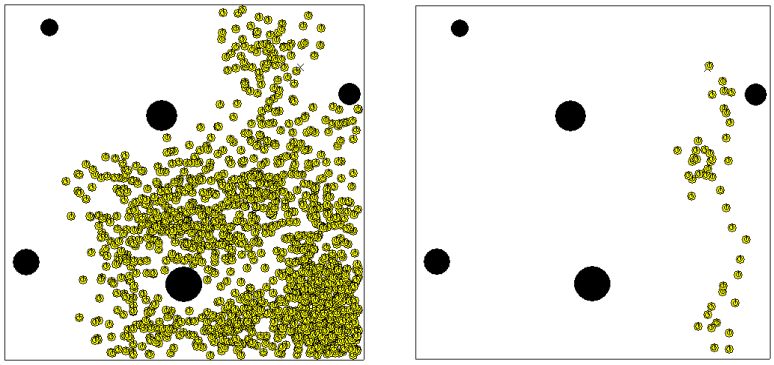

Fig. 6. The positions occupied by the simulated robot the first time that it reaches the goal by reinforcement learning (*left*) and by planning and acting (*right*)

A simulation was run to verify this. The simulated robot was set at the southeast start. The action "selected" was fixed to northwest by suitably changing the simulation program. A sequence of 29 successive predictions was then recorded (see Fig. 7). The results show a quite surprising capacity of the predictor to anticipate the consequences of actions. For example the predictor is capable of coping with noise, is capable of anticipating the appearance of landmarks from behind other landmarks, or the disappearance of them, and the overall chains of predictions are quite similar to the corresponding image sequences (compare the left and right parts of Fig. 7).

Though good, the predictor makes some mistakes (see Fig. 7 and cf. [3]; incidentally, these mistakes are quite interesting). For example it "loses" some landmarks, predicts the appearance of non-existing landmarks, tends to keep fixed images of landmarks at the middle of the scene (this is a strong regularity learned in the world: when the simulated robot moves toward a landmark, the image of it stays still while the images of all other landmarks move to the sides).

This robustness to noise of the predictor probably depends on the neural-networks' generalization property, and on the non-linearity of the output units of the predictor's experts (recall that these are sigmoid units whose output is compared with a 0.5 threshold and made binary accordingly, see above). When the predictor is trained, the images used as teaching output are those that correspond to the views of the world. As a consequence, even when fed with some images corrupted by noise the predictor's output will tend to be close to an image that corresponds to a view of the world since noise will tend to be filtered out by the non-linear output units.

This section is closed with some remarks on the noise robustness of the reactive system (the noise robustness of the planners is comparable). The system is robust with regards to the noise that affects actions and to the noise that causes the pixels of the retina to flip their values: doubling the variance of the former (from 0.01 to 0.02 of variance: recall that the step measures 0.05) or the probability of the latter (from 0.01

to 0.02) does not disturb the system. On the contrary, the system is very sensitive to the compass noise. Doubling it (from 1 degree to 2 degrees of variance) makes the system fall in local minima. Probably the reason of this sensitivity is that the rotation invariance of images is very important in the absence of a rotation-invariant pre-processing and with the simple one-layer neural networks used here.

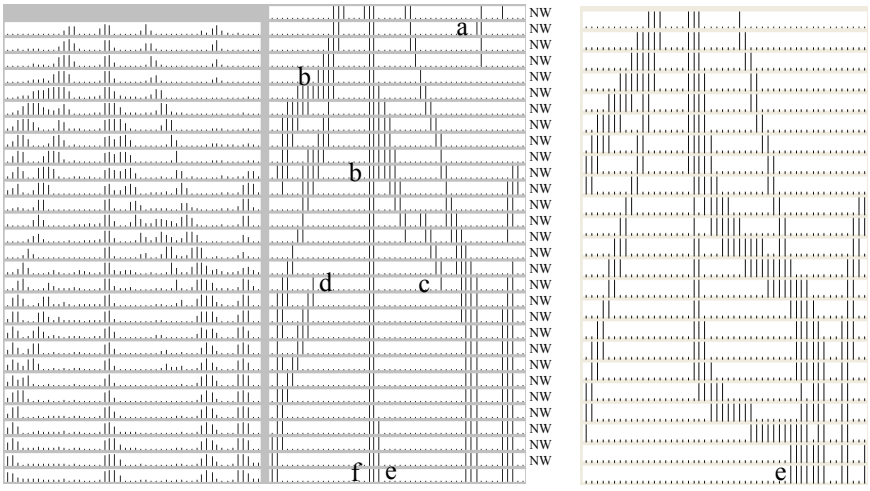

Fig. 7. The graph on the left reports an example of chain of predictions from the southeast start toward northwest (the action selected has been kept fixed to northwest). For each step the continuous output of the predictor (*left part of the graph on the left*), the binary prediction (*right part of the graph on the left*), and the selected action (*column of letters*) are shown. The succession of the predictions is reported from the top to the bottom of the graph. The first binary image corresponds to the image that the simulated robot perceives at the position currently occupied in the world, while the other images are generated by the predictor. *Graph on the right*: true images perceived by the simulated robot while moving from the southeast start toward northwest. It has been reported to allow the evaluation of the quality of the chain of predictions (reported in the left graph). Notice that this graph has been expanded to have a better correspondence with the chain of predictions: this means that the chains of predictions tend to be made up by more steps than the sequences of real images. Notice that the predictor is capable of coping with noise (*a* indicates a noisy activation that is suppressed after some time), is capable of predicting the appearance of a landmarks from behind other landmarks (*b*) and the disappearance of landmarks behind other landmarks (*c*); however, the predictor also produces distorted images, for example it generates non-existing landmarks (*d*) and it has biases, e.g. the prediction chain leads toward the north west landmark and not to the left of it as it should (*e*) probably because landmarks' images perceived at the center of the retina tend to persist (*f*)

4 Bidirectional Planning

We have seen that the forward planner needs a lot of planning cycles to reach the goal the first time. Is there a way to increase the planner's efficiency? One possible solution is to exploit the information given by the goal image. Next sections propose a

bidirectional planner that exploits this information by generating chains of "predictions" both from the current position and from the goal image, and show that this planner has some important advantages in comparison to the forward planner.

4.1 The Architecture and Functioning of Bidirectional Planning

If the variable "BidirectionalPlanning" of the algorithm illustrated in Fig. 3 and Fig. 4 is set at "TRUE", the algorithm implements bidirectional planning. As the forward planner, the bidirectional planner decides if planning or acting on the basis of the measure of confidence at the position currently occupied by the simulated robot (line 7 to 14). The major difference between the two algorithms is that while planning the bidirectional planner generates prediction chains alternately forward from the current position image (line 47 to 55 implement one cycle of forward chain) and backward from the goal image (line 38; line 60 to 69 implement one cycle of backward chain). The length of each backward chain is the same as the last forward chain (line 32 and 42). Forward chains are executed as in forward planning. Backward chains are executed through the "back-predictor" and "back-actor" (Fig. 2).

The back-predictor is a network with the same architecture as the predictor. While the predictor is trained to produce the association $\mathbf{y}_t$, $a_t \rightarrow \mathbf{x}_{t+1}$, the back-predictor is trained to produce the association $\mathbf{y}_t$, $a_{t-1} \rightarrow \mathbf{x}_{t-1}$ (time indexes used backward) i.e. to *remember* (or guess) which *situation* $\mathbf{x}_{t-1}$ *led the system to the current situation* $\mathbf{y}_t$ after executing action a_{t-1} (line 62). Notice that each couple of experts of the predictor and of the back-predictor corresponding to a particular action could have been integrated in one bi-directional network associating $\mathbf{x}_t \leftrightarrow \mathbf{x}_{t+1}$ under action a_t. This has not been done since for simplicity only feed-forward networks have been used.

The back-actor has the same architecture as the actor, and is used to generate actions for the backward chains (the a_{t-1} of the association $\mathbf{y}_t$, $a_{t-1} \rightarrow \mathbf{x}_{t-1}$, see line 60 to 62). Before the tests shown below the back-actor weights are randomly drawn in the interval [-0.001, 0.001], so initially it selects actions randomly. During a back cycle that leads from $\mathbf{y}_t$ to $\mathbf{y}_{t-1}$ (from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$), after the back-actor selects a_{t-1}, the merit of this action is updated according to the same formula used for the actor (see equation 3) and with the usual error $e_{t-1} = (r_t + \gamma V'^{\pi}[\mathbf{y}_t]) - V'^{\pi}[\mathbf{y}_{t-1}]$ (line 67). However, now the merit of the action is updated using $\mathbf{y}_t$ as input for the back-actor (and not $\mathbf{y}_{t-1}$ as for the forward actor). Notice that with this training the back-actor learns to generate actions that lead to states with the *lowest possible evaluation* $V'^{\pi}[\mathbf{y}_{t-1}]$, i.e. states *far* from the goal and visited few times. When backward chains are generated, the actor and evaluator are also updated using e_{t-1}. In particular the actor produces the actions' merit in correspondence to $\mathbf{y}_{t-1}$, and then its weights are updated on the basis of those merits and the action a_{t-1} selected by the back-actor and back-stochastic selector (line 68 and 69). During forward planning and acting, the back-actor is also trained by using e_{t-1} (line 56 to 58 and 79 to 81). To this end, in each forward cycle the back-actor yields the actions' merit $\mathbf{m}_{t-1}$ in correspondence to $\mathbf{y}_t$, and then its weights are updated on the basis of those merits and the action a_{t-1} selected by the actor and stochastic selector for $\mathbf{y}_{t-1}$. The overall functioning of the bidirectional planning algorithm can be summarized as follows. The back-actor learns to yield backward chains that

"escape" from the goal in "straight" lines, hence creating a big area of positive evaluations around the goal. This area is easily "found" by the forward planning chains that, as a consequence, expand the same area toward the position occupied by the simulated robot. At the same time the actor becomes competent in the area where positive evaluations diffuse, and soon gets ready to act efficiently in the world.

Some remarks about backward planning are due. Updating the evaluator when the back-actor is selecting the actions may cause some problems. In fact actor-critic methods require that state evaluations reflect the expected reward averaged over the actions *selected by the current policy* (actor) in that state (notice that one could avoid this problem using "off-policy reinforcement learning methods", cf. [25]). Notwithstanding this, the choice made here should not be impairing because the actor's policy and the back-actor's "back-policy" should be quite similar. In fact: (a) the actor and the back-actor have the same architecture and are trained an equal number of times with the same errors; (b) the probability that the back-actor selects the action a_{t-1} at $\mathbf{y}_t$ ($\mathbf{x}_t$) is similar to the probability that the actor selects the same action at $\mathbf{y}_t$ ($\mathbf{x}_{t-1}$) (yielded by the back-predictor on the basis of $\mathbf{y}_t$ and a_{t-1}) since $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ tend to be perceptually very similar; (c) the direction of the maximum slope of the evaluation gradient field built by the actor and by the back-actor tends to be the same (i.e. toward the goal). Incidentally, notice that these observations suggest that maybe it is possible to integrate the actor and back-actor in a unique network. It remains to be ascertained which are the domains different from navigation where these assumptions still hold.

Backward planning should present two important advantages vs. forward planning. The first advantage is exploration. Updating the evaluations backward from the goal brings immediately to change the evaluations of states close to the goal. On the contrary forward planning starts to update the evaluations only after the goal is encountered the first time. Since the first search of the goal is usually done by random walk (but cf. [30]), that event can take very long to occur (the expected time is exponential in the number of steps separating the start from the goal, cf. [30]). The second advantage is in terms of propagation of evaluations. This is particularly fast if done backward from the goal because newly updated evaluations of states are used to update the evaluations of other states [14] [30].

4.2 Forward and Bidirectional Planning: Results and interpretations

The forward and bidirectional planners have been tested with the scenario illustrated in section 2. The task assigned to the simulated robot was to reach the goal several times each time departing from one of the 12 starting positions showed in Fig. 1 in a sequence, beginning with the northwest one (after the southeast starting position was used, the sequence repeated until the end of the test). The results show that both planners implement genuine taskable planning. In fact they respectively reach the goal for the first time in 245 and 186 moves in comparison to the reactive system that takes 1432 moves (averaged over 10 random seeds). Both planners suitably interleave planning and reactive behavior (Fig. 8 and Fig. 9): when they repeatedly reach the goal from the same start, the performance improves both in terms of planning cycles and

moves; moreover, the experience gathered while planning to reach the goal from a given starting position is transferred to the starting positions close to it (Fig. 8).

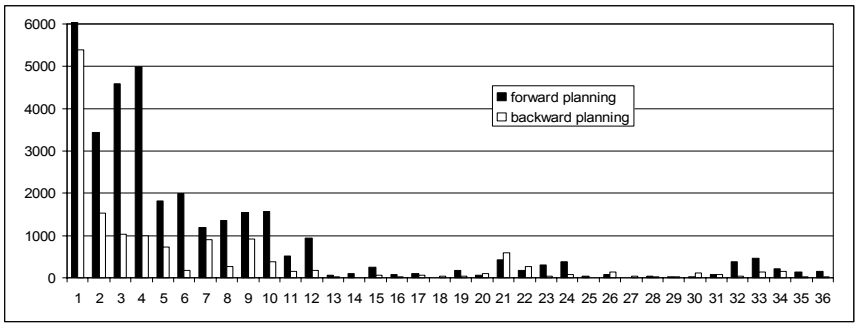

Fig. 8. Number of cycles (*y-axis*) spent planning for each success (*x-axis, 36 successes*) of the forward and bidirectional planners. Average over 10 random seeds. For graphical reasons the vertical axis is cut at 6000 (forward planning took 52,923 cycles to reach the first goal)

If the performance of the planners is compared, the following differences become apparent. The bidirectional planner is much more "goal oriented" than the forward planner. The forward planner spends nearly ten times the planning cycles used by the bidirectional planner (52,923 vs. 5,397 cycles) to reach the goal for the first time. After the first success in the world, bidirectional planning maintains its superiority for the following successes (Fig. 8). This difference is due to the fact that the forward planner takes several cycles to find the goal the first time (18,892 cycles on average). In comparison bidirectional planning is particularly efficient: in 6 tests out of 10 it reaches the goal in the world (for the first time) without having ever reached it in (forward) planning mode.

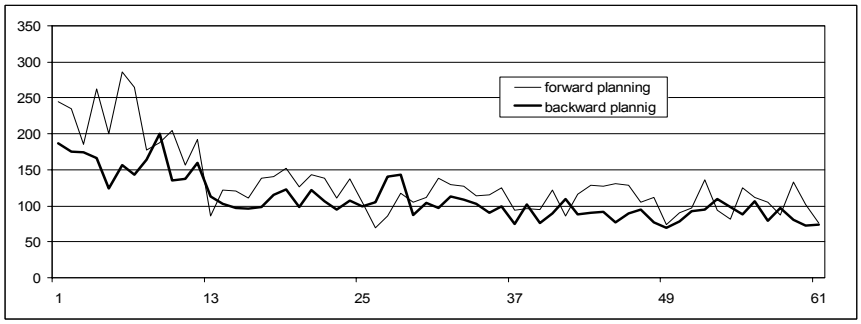

Fig. 9. Number of cycles (*y-axis*) spent acting for each success (*x-axis, 61 successes*) by the forward and the bidirectional planners. Average over 10 random seeds

This efficiency in exploration implies a fast "propagation" of the correct evaluations and policy updating through states. Fig. 10 shows the evaluations produced by the two planners in 20×20 positions of the arena after some cycles of action and plan-

ning. The bidirectional planner yields evaluations much closer to the optimal ones (equal to γ to the power of the number of steps to the goal) than the forward planning.

The bidirectional planner is also more effective than the forward planner in propagating the evaluations back from the goal. Direct observation of the dynamics of the graph of Fig. 10 drawn during the simulation, shows that, at the beginning of the simulation, in the case of the forward planner the evaluations tend to fall again to 0 between a (planned) reaching of the goal and the next. This happens because the goal is reached rarely and the evaluations get close to 0 due to the decay coefficient that, on the average, lowers the "targets" of the evaluations' updating (see formula 3). In the case of the bidirectional planner, positive evaluations are continuously "injected" in the graph from the goal state, and from there rapidly diffuse to contiguous states.

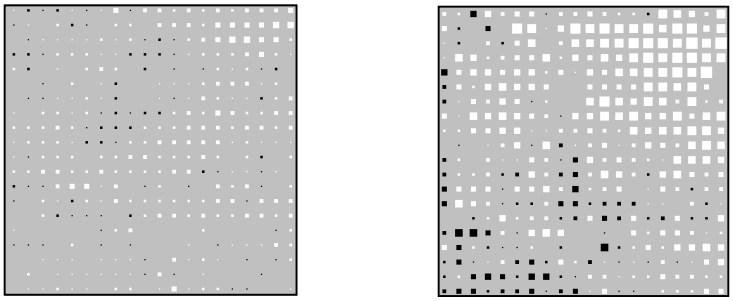

Fig. 10. Evaluations produced by the forward (*left*) and bidirectional (*right*) planners when put in 20×20 different positions of the arena. The area of the *white squares* (positive evaluations) and the *black squares* (negative evaluations) is proportional to the absolute evaluation produced at that position. *Gray* in correspondence to a position indicates an evaluation close to 0. For the forward planner, the evaluations have been recorded at the cycle after which the goal was reached the first time with a chain of predictions. For the bidirectional planner, evaluations were recorded after 13,340 cycles: this is the average number of cycles taken by the forward planner to reach the goal for the first time with a chain of predictions. Negative evaluations, that should not be present since the task does not involve negative rewards, are due to the insufficient number of degrees of freedom of the evaluator and the generalization properties of networks (negative evaluations are more pronounced in the case of the bidirectional planner simply because it is in a more advanced stage of learning in comparison to the forward planner)

5 Related Literature and Novelties of the Paper

The actor-critic part of the planners is similar to the actor critic models implemented with neural networks in [14] and [25]. The only difference is the way the actions are selected stochastically (see equation 2).

The idea of implementing planning as a form of (reinforcement) learning within a model of the world has been proposed with the Dyna models [26] (also compare [5], on "trial-based real-time asynchronous dynamic programming" applied to path finding problems, with the planners presented here). It is important to stress that previous works (e.g. [14] and [26]) used Dyna architectures exclusively to speed up learning,

not to implement genuine taskable planning as done here. The reason is probably that no device like the matcher was used. The idea of generating simulated experiences on the basis of the current policy, called "trajectory sampling", was investigated in [5] and [25] but this is the first time that long prediction chains are generated with neural networks and their properties are investigated. The idea of increasing the depth of the path generated during planning resembles the “iterative deepening search” applied to deterministic problems by problem solving research [12]. However, the algorithms proposed here are new since they deal with stochastic problems.

The forward and bidirectional planning algorithms that control the flow of information between the components of the systems and generate the prediction chains are new. The idea of implementing the predictor (model of the environment) with a feed-forward neural network trained with experience has already been used in [14] and [18]. Notice that these works, as here, use deterministic neural networks to implement a model of a stochastic world. An alternative would have been to use stochastic networks, such as the feed-forward stochastic networks proposed in [17]. In this paper the predictor was trained while the simulated robot randomly explored the world. More sophisticated ways of exploring the world for model building have been proposed, e.g. in [23] and [30]. The idea of using expert networks for the predictor, each specialized to predict the consequences of one action, is proposed and used in [13].

The general idea of planning backward from the goal is widely used in the literature on classic planning [1]. However, its application to stochastic worlds, as done here, implies completely new problems, so the algorithms proposed here are new in this respect. The idea of updating the evaluations backward from the rewarding states has already been investigated within the reinforcement learning literature. In particular [13], [14], [19] and [30] have shown that this is a powerful strategy because state-evaluations are updated on the basis of recently updated evaluations. However, all these works used memory structures to store sequences of states that led to rewarding states in order to use them for iterated “backward” backups. If one wants to use neural networks, this strategy raises the problem of how implementing these memory structures and how using the information stored in them. With this respect, the back-actor, back-predictor and matcher used here are new since they allow generating rewards and states backward from the goal at will. Prioritized sweeping [7] [16], by updating states or state variables whose evaluations would change a lot if updated, often propagates evaluations backwards from states close to the goal.

There are two other important branches of research related to planning with neural networks. One is activation-diffusion planning (see several examples in [15]). These models are based on maps of Kohonen-like units [11]. Each of these units stores a “snapshot” of the world in its weights. While the agent randomly navigates in the world, lateral connections are formed/strengthened between units corresponding to places that are contiguous in space. Planning is implemented by “injecting” activation into the unit corresponding to the goal, allowing it to diffuse through lateral connections with a progressive attenuation, and selecting the actions so as to ascend the activation gradient field formed over the units. The model presented here differs from these models because: (a) places are represented by *patterns of active cells* instead of *patterns of weights*; (b) planning is based on focussed active exploration of the possible actions’ consequences; (b) distributed representations are used allowing one unit

to participate to the representation of several places (while activation diffusion planning needs to use one unit for each place represented).

The second branch of research is represented by neural planners based on gradient descent processes ([22], [28] and [29]). These planners formulate planning problems in terms of differentiable cost functions (and eventually differentiable actions) so they have a limited applicability. Plans are found by minimising these functions. Two other works relevant for the issues tackled here are [18] and [27]. They present two systems that use neural models of the world respectively to improve the performance of an agent controlled by a neural network evolved with genetic algorithms, and to allow a simulated robot to learn the world's regularities at multiple levels of abstraction.

6 Conclusion and Future Work

This paper has presented two new reactive and planning systems implemented with neural networks and inspired by Dyna-PI reinforcement learning methods. These planners present important novelties. Contrary to Dyna architectures, the planners are capable of executing genuine taskable planning using the goal independent information stored in the model of the world. This allows them to reach the goal with few moves from the first time they pursue a goal. This ability, that relies on the capacity to anticipate future states, is of great advantage if experience is costly or risky. Both planners are capable of nicely interleaving planning and action. They can decide to plan when their "confidence" in action, a novel concept introduced in the paper, is low, and to act reactively when they have enough experience about the goal and the current state. In the future, a better measure of confidence will be based on the concept of "entropy" applied to the actions' probabilities [24]. The model of the world is a compound neural network trained with experience to predict the states deriving from the execution of actions. The tests of the planners run with a simulated robot engaged in a stochastic path-finding task have indicated that the chain of predictions generated on the basis of this model of the world are more robust than expected, due to the non-linearity and generalization capabilities of neural networks. Since the model of the world is the core component of each planning system, this interesting result will be further investigated in the future. The bidirectional planner, that exploits the knowledge of the goal to generate "prediction" chains backward from it, has shown to be more efficient in exploring the state space and quicker in propagating the evaluations back from the goal than the forward planner. Unfortunately, these advantages have been obtained at the cost of a rather complex architecture. In the future a simpler planner will be studied that generates forward chains as the forward planner presented here, and updates the goal state's evaluation with a target of $1+\gamma$ when a forward chain terminates without encountering the goal ($1+\gamma$, and not simply 1, will be used since the states encountered one step before reaching the goal are updated on the basis of the undiscounted reward of 1). Given the generalization property of neural networks, this should change the evaluation of several states around the goal state, creating an extended area with positive evaluations. This would facilitate the forward chains in finding the goal state, allowing one to have some advantages similar to those obtained

with the bidirectional planner, but with an architecture as simple as the forward planner's one (see [3] for details).

Acknowledgements

The Department of Computer Science of the University of Essex funded the author's research. Special thanks are expressed to my supervisor Prof. Jim Doran (University of Essex) who encouraged the exploration of the idea of bidirectional planning. The Institute of Cognitive Science and Technologies of the National Research Council of Italy (ISTC-CNR) funded the author's research during the writing up of the paper.

References

1. Allen J., Hendler J., Tate A. (eds.): Readings in Planning. Morgan Kaufmann, Palo Alto Ca. (1990)
2. Arkin R.C.: Behavior-Based Robotics. The MIT Press, Cambridge Ma. (1998)
3. Baldassarre, G.: Planning with neural networks and reinforcement learning. Computer Science Department, University of Essex (2002) Ph.D. Thesis
4. Baldassarre G.: A modular neural-network model of the basal ganglia's role in learning and selecting motor behaviors. Cognitive Systems Research. 3 (2002) 5-13
5. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artificial Intelligence. 72 (1995) 81-138
6. Brooks R.A.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation. 2 (1986) 14-23
7. Dearden R.: Structured prioritized sweeping. In: Proceedings of the Eighteenth International Conference on Machine Learning. (2001) 82-89
8. Fikes R.E., Nilsson N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence. 2 (1971) 189-208
9. Haykin S.: Neural Networks: A Comprehensive Foundation. Prentice Hall, Upper Saddle River N.J. (1999)
10. Jaakkola T., Singh S.P., Jordan M.I.: Reinforcement learning algorithm for partially observable Markov decision problems. In: Tesauro G., Touretzky D.S., Leen T.K. (eds.): Advances in Neural Information Processing Systems 7. The MIT Press, Cambridge Mass. (1995) 345-352
11. Kohonen T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics. 43 (1982) 59-69
12. Korf, R.E.: Optimal path finding algorithms. In: Kanal L.N., Kumar V. (eds.): Search in Artificial Intelligence. Springer-Verlag, Berlin (1988) 223-267
13. Lin L.-J., Mitchell T.M.: Memory approaches to reinforcement learning in non-Markovian domains. Carnegie Mellon University (1992) Technical Report CMU-CS-92-138

14. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning. 8 (1992) 293-391
15. Meyer J.-A., Berthoz A., Floreano D. (eds.): From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior. The MIT Press, Cambridge Ma. (2000)
16. Moore A.W., Atkeson C.G.: Prioritised sweeping: Reinforcement learning with less data and less real time. Machine Learning. 13 (1993) 103-130
17. Neal R.M.: Bayesian learning for neural networks. Springer-Verlag, Berlin (1996)
18. Nolfi S., Elman J.L., Parisi D.: Learning and evolution in neural networks. Adaptive Behavior. 3 (1994) 5-28
19. Reynolds S.I.: Experience stack reinforcement learning for off-policy control. University of Birmingham (2002) Technical report CSRP-02-1
20. Ross S.: Introduction to stochastic dynamic programming. Academic Press, New York (1983)
21. Russell S., Norvig P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs N.J. (1995)
22. Schmidhuber J., Wahnsiedler R.: Planning simple trajectories using neural subgoal generators. In: Meyer J.-A., Wilson S.W. (eds.): From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior. The MIT Press, Cambridge Ma. (1992) 196-202
23. Schmidhuber J.: Artificial curiosity based on discovering novel algorithmic predictability through coevolution. In: Angeline P., Michalewicz X., Schoenauer M., Yao X., Zalzala Z. (eds.): Congress on Evolutionary Computation. IEEE Press, Piscataway N.J. (1999) 1612-1618
24. Shannon C.E.: A mathematical theory of communication. The Bell System Technical Journal. 27 (1948) 623-656
25. Sutton R.S., Barto A.G.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge Mass. (1998)
26. Sutton R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceeding of the Seventh International Conference on Machine Learning. Morgan Kaufmann, San Mateo Ca. (1990) 216-224
27. Tani J., Nolfi S.: Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems. Neural Networks. 12 (1999) 1131-1141
28. Tani J.: Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective. IEEE Transactions in System, Man and Cybernetics, Part B. 26 (1996) 421-436
29. Thrun S. B., Moller K., Linden A.: Planning with an adaptive world model. In: Tourtezky D. S., Lippmann R. (eds.): Advances in Neural Information Processing Systems 3. Morgan Kaufmann, San Mateo Ca. (1991) 450-456
30. Thrun S.: Efficient exploration in reinforcement learning. Carnegie-Mellon University (1992) Technical Report CMU-CS-92-102
31. Widrow B., Hoff M.E.: Adaptive switching circuits. IRE WESCON Convention Record, Part IV. (1960) 96-104