A PLANNING MODULAR NEURAL-NETWORK ROBOT FOR ASYNCHRONOUS MULTI-GOAL NAVIGATION TASKS

Gianluca Baldassarre
University of Essex, Colchester, United Kingdom, gbalda@essex.ac.uk

Abstract
This paper focuses on two planning neural-network controllers, a "forward planner" and a "bidirectional planner". These have been developed within the framework of Sutton's Dyna-PI architectures (planning within reinforcement learning) and have already been presented in previous papers. The novelty of this paper is that the architecture of these planners is made modular in some of its components in order to deal with catastrophic interference. The controllers are tested through a simulated robot engaged in an asynchronous multi-goal path-planning problem that should exacerbate the interference problems. The results show that: (a) the modular planners can cope with multi-goal problems allowing generalisation but avoiding interference; (b) when dealing with multi-goal problems the planners keeps the advantages shown previously for one-goal problems vs. sheer reinforcement learning; (c) the superiority of the bidirectional planner vs. the forward planner is confirmed for the multi-goal task.

1. Introduction
This paper focuses on a simulated robot (Figure 1) engaged in a stochastic shortest-path problem [3]. In this kind of problem the robot has to find the shortest (e.g. in terms of number of states visited) path from a start state to a goal state in a stochastic world.
Reinforcement-learning algorithms [13] are capable of finding an action policy by using a trial-and-error strategy directly implemented in the world. When a model of the world is available, stochastic dynamic programming methods are capable of generating action policies by using the model. The policies are then executed in the real world, hence implementing a form of planning. Stochastic dynamic programming methods are based on the generation of a gradient field of "evaluations" associated with the states of the problem and creating an action policy that ascends the gradient field towards the goal state.

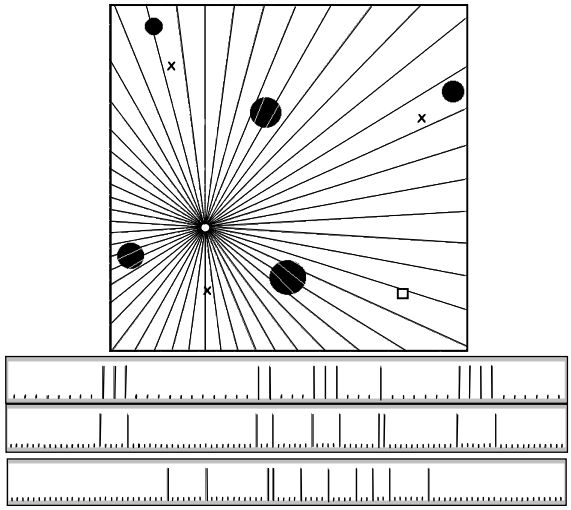

Figure 1: *Top:* the scenario of the simulations containing three goals (north-west, east, and south-west, marked with *x*), a start (*white square*), five landmarks (*black circles*), the scope of the organism's 50 visual sensors (delimited by the *rays*), and the robot (*white circle* at origin of rays). *Bottom:* the pattern of the visual sensors, its re-mapping into contrasts, and an example of goal (contrast pattern, for the south west goal).

Sutton [12] has integrated planning based on dynamic programming and reinforcement learning into a class of architectures called "Dyna architectures" (from "dynamic programming"). The basic idea of Dyna architectures is to have a reinforcement learning architecture that is trained both in the real world and through a model of the world used to generate "simulated" extra experience.
Baldassarre [2] has proposed two planning controllers inspired to Dyna-PI architectures, a Dyna architecture based on actor-critic reinforcement-learning methods [13] ("PI" stands for "policy iteration", see later). Actor-critic methods use two memory structures: one to store

the evaluation function and the other one to store the action policy. The algorithms proposed in [2] are capable of operating in "reinforcement learning mode" or "planning mode". While planning the algorithms execute a sequence of forward "explorations" from the current state (forward planner) or both forward from the current state and backward from the goal (bidirectional planner) within the model of the world. During these explorations the state evaluations and the action policy are updated. The action probabilities are used to build a measure of the robot's "confidence", and to switch between acting and planning mode.

Baldassarre [1] has proposed a modularised version of the basic neural-network actor-critic architecture capable of coping with multiple goals assigned to the robot asynchronously (the robot has to pursue different goals in different times). In particular the critic is modularised on the basis of the "mixture of experts" neural network model [6] (suitably adapted to cope with the reinforcement learning evaluation problem) and the actor is modularised on the basis of a novel hierarchical architecture. The simulations with this architecture showed that it is capable of copying with potential catastrophic interference caused by the multi-goal task. This is a crucial issue for the scalability of neural architectures. Catastrophic interference is the well-known phenomenon that affects neural networks: experience learned by dealing with a particular goal/problem is easily disrupted by the experience learned by dealing with another goal/problem [11]. The architecture proposed in [1] copes with catastrophic interference through "emergent functional modularity" (the structure of the modules is hardwired, but their function is emergent, cf. [4]).

The novelty of this paper is that the planning architectures of [2] are integrated with the modular architecture proposed in [1], and that the performance of the reinforcement learning and the two planners is tested and compared with an asynchronous multiple-goal task. This task highlights whether controllers are affected by catastrophic interference because different goals are pursued at different times. The test is important because catastrophic interference is expected to have a stronger effect on the neural planning controllers proposed in [2] than on the reactive controller. In fact while planning the planning controllers execute a long deep updating of the weights to reach one particular goal before passing to another goal. This was expected to worsen the negative effects of catastrophic interference (cf. [11]).

In particular, the simulations run with the multi-goal task have verified if and why: (a) the modular neural planners are capable of copying with interference; (b) the neural planners outperform the (corresponding) reactive controllers; (c) the bidirectional planner outperforms the forward planner in terms of planning and acting cycles needed to achieve the goals.

Section 2 presents the scenario of the simulations and the reinforcement learning and planning algorithms. Section 3 illustrates the results of the simulations, their interpretation, and the drawbacks of the controllers. Finally section 4 summarises the results and discuss if the possible application of the algorithms to real robots.

2. Scenario of Simulations and Controllers

This section illustrates the simulation scenario used to test the controllers, and the reactive and planning neural components of the architectures. The planning components are illustrated in two different subsections to highlight the differences between the forward and the bidirectional planners.

Simulated Scenario and Robot

The scenario is shown in Figure 1. It is a square arena with sides measuring 1 unit, inside of which there are 5 circular landmarks/obstacles.

The robot can see the landmarks with a one-dimension horizontal retina of 360 degrees. The retina is made up of 50 units (vector $\boldsymbol{x}$). Each unit x_i activates with 1 if a landmark is in its scope, with 0 otherwise, and is affected by noise (0.01 probability of flipping). The signals coming from the retina are always aligned with the magnetic north through a "compass" affected by Gaussian noise (0 mean, 1 degree variance). Before being sent to the controller, these signals are re-mapped into a vector $\boldsymbol{y}$ of 100 binary units representing the image "contrasts". Two contiguous retinal units activate (with 1) one contrast unit y_j if they are respectively on and off, another contrast unit if they are respectively off and on, no contrast units if they are both on or both off (cf. Figure 1). This re-mapping implements an expansion of the input space that allows the controller to work properly with simple two-layer networks in the scenario considered here. Notice that the robot has a limited perception of the world's current state. This can raise difficulties not directly dealt with here (cf. [5] on "partially observable Markov decision problems").

At each cycle of the simulation the robot has to select one of eight actions, each consisting of a 0.05 step in one of eight directions aligned with the magnetic north (north, north-east, etc.). The outcome of these actions is affected by Gaussian noise (0 mean, 0.01 variance). If the robot moves against the arena's boundaries or the obstacles, it bounces back.

The robot's task is to reach three different goal positions in the arena. At the beginning of the simulation the robot is set at the start position and has to reach the east goal. Then each time the robot reaches a goal, one of the other two goals is assigned to it at random until the simulation stops.

Figure 2 shows both the reinforcement learning and the planning components of the robot's neural controller. Now these are analyzed in detail.

Reinforcement learning

This part of the model is slightly different from the actor-critic models, implemented with neural networks, proposed in [13]. In general terms, the actor yields a

stochastic action-selection policy, and the evaluator evaluates the states of the world in terms of expected future rewards achievable with the current actor's policy. The evaluator improves the quality of its evaluations by experiencing the rewards through a supervised learning algorithm, while the actor improves the action-selection policy by increasing the probabilities of actions that cause the robot to ascend the gradient field of evaluations (policy iteration).

Now the single reinforcement-learning components are analysed. The "matcher" is a hand-designed neural network whose role is to internally generate the reward signal r_t. The matcher yields 1 as output when the goal and the input (contrasts) share more than 94% of bits with the same values, and 0 otherwise (cf. [2] for details). Notice that while planning the matcher plays the role of the model of the world concerning rewards.

The "actor", a modular network composed of 6 "expert" networks and 1 "gating network", implements the robot's action-selection policy. Each expert is a two-layer feed-forward neural network that gets goal and visual contrasts as input, and has 8 sigmoidal output units that locally encode the actions. To select one action, the activation m_l (interpretable as "action merit") of the output units is sent to a stochastic selector where a stochastic "winner-takes-all competition" takes place. The probability P that a given action a_l becomes the winning action a_w (to execute) is given by:

$$P[a_l = a_w] = m_l / \Sigma_f m_f \quad (1)$$

This formula has been chosen over the more popular, but also more complex, softmax formula [13] because it led to similar results. The role of the gating network is to select an expert that, in its turn, selects the actions to be executed. This is done with another "winner-takes-all competition" analogous to the previous one, but this time involving the experts instead of the actions (it is based on the "experts' merit", i.e. the activation u_k of the gating-network output units).

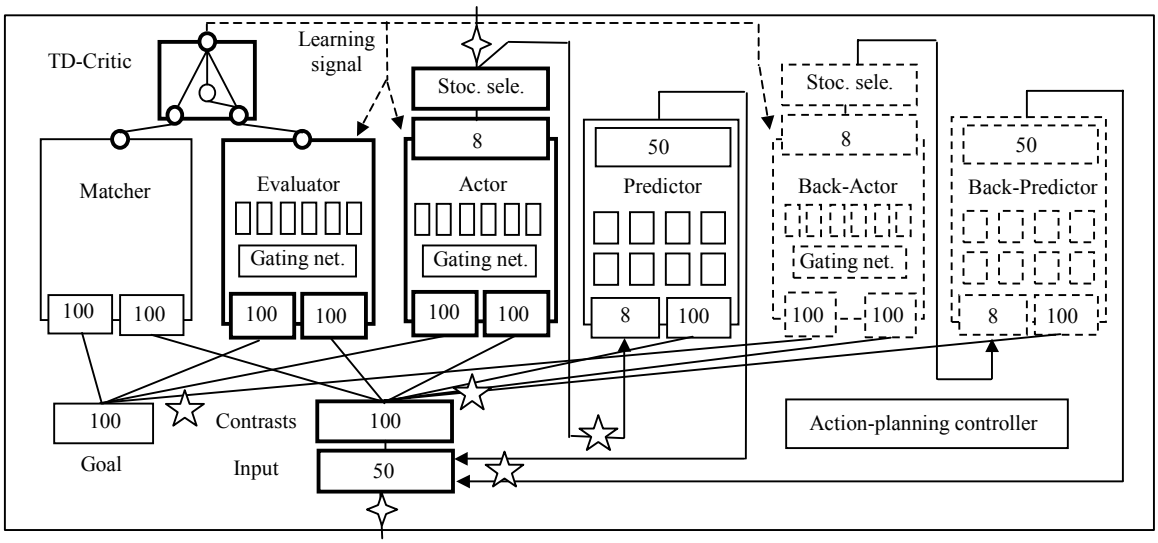

Figure 2: The controller of the robot. Networks with a *bold*, *thin* and *dashed border* implement reinforcement learning, forward planning, and backward planning respectively. *Arcs* and *arrows* indicate forward and backward connections that "copy" a pattern from one layer to another. The *four* and *five spike stars* indicate the channels respectively set open and close 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" (which together with the TD-critic makes up the "critic") is a "mixture of experts network" composed of 6 experts and 1 "gating network", suitably modified to cope with the reinforcement-learning evaluation problem. See [6] for the details of this architecture, and for the mathematical justification of the training algorithm described later. Each expert is a two-layer feed-forward neural network that gets goal and visual contrasts as input. With its linear output unit, the evaluator yields the estimation $V'^{\pi}[\mathbf{y}_t]$ of the correct evaluation $V^{\pi}[\mathbf{y}_t]$ of the current state $\mathbf{y}_t$ (contrast pattern). $V^{\pi}[\mathbf{y}_t]$ is defined as the expected discounted sum of all future reinforcements r, given $\mathbf{y}_t$ and the current action-selection policy π expressed by the actor:

$$V^{\pi}[\mathbf{y}_t] = E[\gamma^0 r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \ldots] \quad (2)$$

where $\gamma \in (0, 1)$ is a "discount factor", set to 0.95 in the simulations, and E is the mean operator. In order to compute $V'^{\pi}[\mathbf{y}_t]$ the output of the experts is weighted and summed:

$$V'^{\pi}[\mathbf{y}_t] = \Sigma_k [v_k g_k] \quad (3)$$

where v_k is the output of the expert k, and the weight g_k is computed as the "softmax activation function" of the activation of the output units o_k of the gating network:

$$g_k = exp[o_k] / \Sigma_f [exp[o_f]] \quad where: \Sigma_k g_k = 1 \quad (4)$$

The "TD-critic" is a neural implementation of the computation of the "temporal-difference error" e_t ("learning signal" in Figure 2) defined as [13]:

$$e_t = (r_{t+1} + \gamma V'^{\pi}[\mathbf{y}_{t+1}]) - V'^{\pi}[\mathbf{y}_t] \quad (5)$$

Each evaluator's expert has a *specific error* defined as:

$$e_{kt} = (r_{t+1} + \gamma V'^{\pi}[\mathbf{y}_{t+1}]) - v_k[\mathbf{y}_t] \quad (6)$$

Each evaluator's expert is trained on the basis of the *expert's* learning signal, which assumes the role of error (in the estimation of $V'^{\pi}[\mathbf{y}_t]$) in a supervised learning algorithm. The weights of the experts are updated so that their estimation $v_k[\mathbf{y}_t]$ tends to be 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 new estimation $V'^{\pi}[\mathbf{y}_{t+1}]$. The formula (a modified Widrow-Hoff rule [15]) used to update the weights of each expert is:

$$\Delta w_{kj} = \eta \, e_{kt} \, y_j \, h_k \qquad (7)$$

where η is a learning rate (set to 0.02), w_{kj} is a weight of the expert k, and y_j is the activation of the goal and contrast units at time t (evaluator's input). h_k (absent in the Widrow-Hoff rule) is the "updated" contribution of the expert to the global answer $V'^{\pi}[s_t]$, and is defined as:

$$h_k = g_k \, c_k \, / \, \Sigma_f [g_f \, c_f] \qquad \text{where: } \Sigma_k \, h_k = 1 \qquad (8)$$

where c_k is a measure of the "correctness" of the expert k defined as:

$$c_k = exp[-0.5 \, e_{kt}^2] \qquad (9)$$

The gating network weights z_{kj} are updated to increase the weight g_k of the experts that had low error:

$$\Delta z_{kj} = \xi \, (h_k - g_k) \, y_j \qquad (10)$$

where ξ is a learning rate set to 0.2 in the simulations. This algorithm causes the experts to specialise in the different regions of the goal-contrast space ([6] for details). Notice that ξ is higher than η. This has been found to be a necessary condition for the controller to work. With $\xi = \eta = 0.02$ the experts did not specialise and interference between different goals prevented the convergence of the algorithm.

The actor is trained according to the TD-critic's learning signal e_t. Here this signal is a measure of the actor's capacity to select actions that bring the robot to new states with an evaluation higher than the average evaluation experienced previously departing from that same state. The updating of the action merit of the "winning expert" (and only this) is done by changing the weights of the neural unit corresponding to the selected action a_w (and only this) as follows:

$$\Delta w_{wj} = \zeta \, e_t \, (4 \, m_w \, (1 - m_w)) \, y_j \qquad (11)$$

where ζ is a learning rate, set to 0.02 in the simulations, and $(4 \, m_w \, (1 - m_w))$ is the derivative of the sigmoidal function multiplied by 4 to homogenise the size of the learning rates of the actor and the (linear) evaluator. The weights of the gating network (only those of the winning expert's unit) are updated by using the merit u_w and error e_t ($\zeta = 0.02$):

$$\Delta w_{wj} = \zeta \, e_t \, (4 \, u_w \, (1 - u_w)) \, y_j \qquad (12)$$

At the beginning of the simulation the weights of the evaluator and actor are randomised 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 around 0.5 (0.125). This implies that initially the robot explores the environment randomly, and then it starts to shape the evaluations on the basis of the rewards, and the probabilities on the basis of the evaluations.

Forward Planner

The components added to the reactive-learning model to obtain the forward planner are now explained. The "predictor" (robot's "model of the world") is composed of 8 feed-forward two-layer networks ("experts") with sigmoid output units, each corresponding 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 (each sigmoid unit's output is squashed to 0 if below 0.5, to 1 if above). A hand-designed selector chooses the expert corresponding to the selected action to yield the output of the predictor itself. The experts are trained while the robot navigates randomly in the environment for 200,000 cycles (this brings the mean square error per unit to about 0.24). This training is done before the robot faces the task. 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 teaching output to train the expert corresponding to the action with a Widow-Hoff rule [15] (cf. [8] and [9]). Notice that, because of its architecture, the predictor yields deterministic predictions that tend to be the average of the $\mathbf{x}_{t+1}$ observed after each $\mathbf{y}_t$. This is a simplification given that a correct model of the stochastic world should yield stochastic predictions.

The "action-planning controller" is a hand-designed algorithm (Figure 3) that controls the flow of information among the different components of the whole system when the robot is acting and planning forward and backward. Notice that the forward planner is obtained by setting the variable "*OnlyForward-Planning*" (Figure 3) to "*TRUE*", while the bidirectional planner is obtained by setting it to "*FALSE*". Now the forward planner is explained.

The forward planner can be in either planning or acting mode. The action-planning controller decides the robot's mode on the basis of its "confidence" (the highest of the actions' probabilities). If the confidence is above a threshold the robot acts in the world and the predictor is not used, otherwise it is used, together with the matcher, to simulate experience. While planning the threshold slowly decreases (cf. variable "*Decay*") so that the robot changes position after some time spent planning. This prevents the robot from getting stuck in places where it does not succeed to become "confident" enough. When acting the threshold is increased up to a maximum (cf. variables "*MaxConfThresh*" and "*Gain*", Figure 3).

When the robot is forward planning the actor and critic function and learn in the same way they do when acting in the real world. In particular, in a cycle of planning these events take place: the matcher, the actor, and the critic take the signal from the sensors/contrasts activated by the predictor (or from the world if it is the first step of a "simulated walk", see later) as input; the matcher returns the reward signal, the evaluator returns the evaluation of the input pattern, the TD-Critic returns the error e, and the actor (stochastic selector) yields the actions' merits (probabilities) on the basis of which

action is selected; the selected action and the input pattern are sent to the predictor, which in turn yields the predicted input using the network corresponding to the action. At this point a new cycle can take place, yielding a new reward, a new evaluation, and a new error *e*; this new error is used to train the actor and critic to evaluate and act in correspondence to the old input pattern.

```
IF(NewGoalHasBeenAssigned)
  MaxStepsPlan := 1;
  ConfThresh := MaxConfThresh;
  ForwardPlanning := TRUE;
  StepPlan := 0;
  Planning := TRUE;
IF(InputOutputRealWorld)
  IF(Confidence < ConfThresh)
    Planning := TRUE;
  ELSE
    {Planning := FALSE; ConfThresh :=
      MIN(MaxConfThresh, ConfThresh + Gain);}
IF(Planning)
  InputOutputRealWorld := FALSE;
  StepPlan++;
  ConfThresh := ConfThresh - Decay;
  IF(ForwardPlanning)
    IF(GoalReached OR StepPlan = MaxStepsPlan)
      IF(StepPlan = MaxStepsPlan)
        MaxStepsPlan++;
      ELSE
        {MaxStepsPlan :=
          MIN(MaxStepsPlan, StepPlan * 2);}
      ForwardPlanning := FALSE;
      ForwardSteps := StepPlan;
      GoalAsInput := TRUE;
      StepPlan := 0;
      IF(OnlyForwardPlanning)
        ForwardPlanning:= TRUE;
        GoalAsInput := FALSE;
        InputOutputRealWorld := TRUE;
  ELSE
    GoalAsInput := FALSE;
    IF(StepPlan = ForwardSteps)
      ForwardPlanning := TRUE;
      InputOutputRealWorld := TRUE;
      StepPlan := 0;
```

Figure 3: Pseudo-code of the planning-acting controller algorithm (executed at each cycle after the actor's activation). Assignment operator: ":="; 1 unit increment operator: "++". The parameters are set as follows: *Decay* = 0.000001, *Gain* = 0.01, *MaxConfThresh* = 0.15

Given a goal, if in forward planning mode the robot executes several planning cycles in a sequence. Each sequence is a sort of "simulated walk" that starts from the current input pattern, continues through a succession of states (predictions yielded by the predictor), and terminates either when the goal is reached or when the sequence is longer than a certain number of steps (*MaxStepsPlan*). This number is increased if the simulated walk fails to reach the goal (similarly to an "iterative-deepening search" [7]). When success is achieved, the number is decreased to focus exploration around the start and goal.

Bidirectional Planner

The bidirectional planner generates simulated walks alternatively forward from the current input and backward from the goal. The length of each backward walk is the same as the forward walk (the algorithm resembles a "bidirectional iterative-deepening search" [7]). Forward walks are executed as in the forward planner. Backward walks are executed through the "back-predictor" and "back-actor".

The back-predictor is a network with the same architecture as the predictor. While the predictor is trained to produce the association $\boldsymbol{y}_t$, $a_t \rightarrow \boldsymbol{x}_{t+1}$, the back-predictor is trained to produce the association $\boldsymbol{x}_t$, $a_{t-1} \rightarrow \boldsymbol{y}_{t-1}$ (time indexes used backward) i.e. to remember/guess what the situation was that brought the robot to the current situation after executing a particular action. Notice that each predictor's expert and back-predictor's expert corresponding to a particular action could be integrated in one bidirectional network associating $\boldsymbol{x}_t \leftrightarrow \boldsymbol{x}_{t+1}$ under action a_t.

The back-actor has the same architecture as the actor, and is used to generate actions for the simulated backward walks (the a_{t-1} of the association $\boldsymbol{x}_t$, $a_{t-1} \rightarrow \boldsymbol{y}_{t-1}$). Before the tests described later the back-actor weights are randomly drawn from the interval [-0.001, +0.001], so initially it selects actions at random. During a back walk cycle that derives $\boldsymbol{y}_{t-1}$ from $\boldsymbol{y}_t$ the merit of the action selected is updated according to the same formula used for the actor and with the usual error of equation (5). Notice that this training induces the back-actor to generate actions that lead to states with the *lowest* possible evaluation $V^{\pi}[\boldsymbol{y}_{t-1}]$, i.e. states far from the goal and visited few times. During the backward walks the actor and evaluator are also updated (using e_t). In particular after each cycle the actor yields the actions' merit in correspondence to $\boldsymbol{y}_{t-1}$, and then its weights are updated in correspondence to those merits and the action a_{t-1} selected by the back-actor. During forward planning and acting, the back-actor is also trained by using e_t. To this purpose after each forward cycle the back-actor yields the actions' merit in correspondence to $\boldsymbol{y}_{t+1}$, and then its weights are updated in correspondence to those merits and the action a_t selected by the actor with $\boldsymbol{y}_t$. The overall functioning of the backward planner can be summarised as follows. With training the back-actor learns to yield backward walks 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 actor's forward walks that, as a consequence, progressively expand the area itself toward the start. At the same time the actor becomes competent in the whole area where positive evaluations diffuse. Cf. [2] for a deeper analysis of these algorithms and their applicability limitations.

3. Results, Interpretations, and Limits of the Controllers

The reinforcement learning, forward planner and bidirectional planner have been tested with the multi-goal task described in the previous section. 10 simulations with different random seeds were run for each algorithm for a sequence of 2000 achievements of the goals. For all the three conditions 7 out of 10 simulations were successful. In these simulations the

critic used three different experts to encode the three different evaluation gradient fields corresponding to the three goals (in each position the evaluator had a probability above 99% of selecting the same expert). Figure 4 shows the gradient fields relative to 2 goals (1 of the 7 successful seeds).

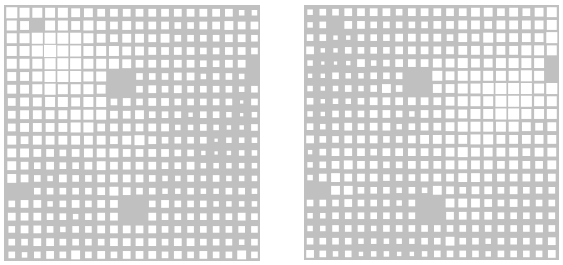

Figure 4: Evaluation gradient fields for two of the three goals (north-west and east goal). For each goal the robot was set in 20×20 different positions of the arena, and the evaluator's output for that position was measured. Each *white square* of the graph is drawn in a position corresponding to the position in the arena where the evaluation was measured. The area of each square is proportional to the evaluation level. *Empty squares* correspond to the landmarks. The gradient fields are higher in correspondence to the goals

5	2	5	5	5	3	5	5	5	5	5	3	3	3	3	3	2	3	3	4
2	2	0	5	5	5	5	3	5	5	5	5	2	1	5	5	2	2	3	3
2	2	2	5	5	5	5	5	5	5	5	5	5	5	2	5	5	5	2	3	3
2	2	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	3	2	3	3
5	2	2	2	5	5	5	5	5	5	5	2	2	5	5	5	2	6	6	0
2	2	2	2	2	5	5	3	0	0	2	2	2	2	3	3	3	3	3	0
2	2	2	2	2	5	5	1	0	0	2	2	3	3	3	3	3	4	4	
2	2	2	2	2	3	3	1	1	5	1	3	3	3	3	4	4	4	1	2
1	2	3	2	2	2	3	1	1	3	3	3	4	2	4	4	4	2	1	2
1	2	3	1	2	2	3	1	3	3	2	4	4	3	4	4	2	1	2	3
1	2	1	1	2	2	2	3	3	2	2	2	5	1	4	1	1	1	3	3
5	2	3	3	2	2	2	3	3	5	5	5	3	5	1	1	1	1	2	5
3	5	3	3	2	2	2	2	2	5	5	5	3	3	1	3	1	1	5	1
5	3	3	2	2	2	2	2	3	3	5	3	3	2	6	3	5	1	1	1
0	0	3	2	2	2	2	5	5	3	3	5	5	3	4	1	1	1	2	3
1	2	2	2	2	2	2	2	2	0	0	2	4	2	3	1	1	3	3	3
3	2	2	2	2	1	2	2	2	0	0	3	3	2	5	1	1	1	3	3
2	2	3	1	1	1	1	1	2	2	3	2	2	2	1	5	1	5	3	3
1	2	2	1	1	1	1	1	1	2	2	1	1	2	1	5	5	4	3	3
1	1	2	3	1	1	5	1	1	2	2	2	1	1	1	1	1	5	3	5

70 120 89 10 87 3

Figure 5: Actor's expert with maximum probability in 20×20 positions of the arena in correspondence to the north-west goal (the position of the goal is marked with a *square*). The small numbers indicate the expert with the highest probability in the position occupied by the number itself. The big numbers summarise the frequencies of use of the 6 experts in the whole arena.

The 3 simulations that failed did so because the critic employed the same expert to yield the evaluations related to two different goals. As a consequence the evaluation gradient field had two peaks, the actor was trained to go to both peaks, and the resulting behaviour was dithering. This fact shows that in the task considered here the specialisation of the evaluator's experts for the different goals is crucial for the correct functioning of the architecture (cf. [1])

In the 7 successful simulations, the actor learned to reach the goals in few steps from any point of the arena (see Figure 6, explained later). The function of the actor does not seem to require a precise specialisation as in the case of the evaluator: more than one expert is used to achieve one goal, and the same expert is used to achieve many goals. For example Figure 5 shows which actor's expert has the highest probability of being selected in 400 positions of the arena for the north-west goal. Clearly different areas of the arena tend to cause the activation of different experts (cf. [1] and [4]). To summarise, the emergent functionality of the modules within the critic and the actor (strong in the former and partial in the later) allow all the three controllers to cope successfully with the asynchronous multi-goal task, keeping the interference problems under control.

For each simulation the number of cycles per success (achievement of the assigned goal) was measured for each reached goal and then plotted against the cumulated cycles. Figure 6 shows this measure for the three algorithms (averaged over the 7 successful random seeds). Several relevant facts become apparent from these simulations, and confirm the results previously obtained with single goal tasks (cf. [1] and [2]). The comparison between the performance of planning vs. sheer reinforcement learning shows that planning allows the robot to reach the goals with improved efficiency from the very first time each goal is assigned to the robot (cf. Table 1; see later for the reason because the second goal is reached with more steps than the first goal). Moreover in all the three cases when the robot pursues the same goal several times the performance improves (cf. again Figure 6). This happens because the information collected through planning and real experience merges suitably and incrementally in the weights of the evaluator and actor. This is a typical strength of the Dyna architectures [12].

Table 1: Number of cycles taken by the three algorithms to reach the first and second goal the first time it is assigned (averaged over 20 seeds).

	Reinforcement learning	Forward planner	Bidirectional planner
1st goal	719	286	199
2nd goal	1407	420	297

With regards to the comparison of the forward planner and the bidirectional planner, Figure 6 shows that before convergence the bidirectional planner outperforms the forward planner in terms of cycles taken to reach the goals. One reason for this is that the forward planner spends more cycles planning than the bidirectional planner (see later) and that while planning the robot executes some actions (cf. algorithm in Figure 3). Another possible reason is that the bidirectional planner focuses exploration and learning not only around the current position (as in the case of the forward planner),

but also around the goal. Direct observation of the robot's behaviour has shown that this area, where the robot has to move to the specific position corresponding to the goal, is a particularly difficult part of the task (consider that when the robot is very far from the goal, about 50% of the moves take it towards the goal, while when very close to the goal only 12.5% i.e. 1 in 8 moves take it to the goal). So when the robot is confident enough at the current state and starts to act, its actual competence for the area near the goal is higher in the case of the backward planner than in the case of the forward planner. Further investigation should verify this explanation and check if this result holds for problem domains different from navigation.

Figure 7 shows the number of planning cycles per success taken by the two planning algorithms for the first 36 goals reached (averaged over 24 successful seeds out of 32: many simulations were run to control noise). A relevant result is that the planning cycles fall close to 0 when the robot experiences the same goals several times. The explanation of this result is that when the same goal is encountered several times, the "confidence" associated with it increases over the confidence threshold, so that planning is no longer required and the goal is achieved reactively.

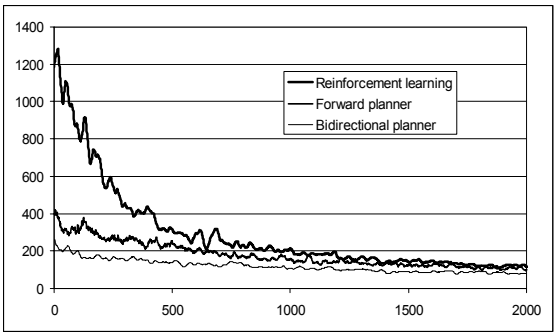

Figure 6: Performance of the three algorithms (average over 7 random seeds per algorithm). *Y-axis*: number of cycles per success (forward moving average over 20 successes). *X-axis*: successes.

Figure 7 also shows that the bidirectional planner outperforms the forward planner in terms of number of cycles spent planning before reaching the goals. This is caused by three factors. The first depends on the way the two algorithms explore the model of the world. The forward planner spends a lot of time searching for the goal unsuccessfully given that the search is a random walk, while the bidirectional planner "finds" the goal as soon as it starts to plan. So contrary to the bidirectional planner, the forward planner wastes a lot of planning cycles before starting to update the evaluations and, consequently, update the action policy (cf. [14]). The second factor is that the bidirectional planner is more efficient in propagating the evaluations backward from the goal to the other states than the forward planner is. In fact it updates the evaluation of each state on the basis of the evaluation of a state that has just been updated (cf. [2]).

The third factor involves the multi-goal nature of the task. After enough confidence is gained and the first goal is reached, the action probabilities are quite biased in favour of that goal. When the goal changes, half actor's input pattern is changed according to the new goal (cf. Figure 2). Given that the experts are not yet specialised for the different goals and that in general the actor is responding to the second goal mostly with the same weights used for the first goal, the actions' probabilities are still quite biased in favour of the first goal. This implies that the random walk used to explore the model of the world is biased *away* from the second goal. Figure 7 shows that the forward planner spends nearly a double number of cycles planning to reach the second goal than to reach the first goal, while the bidirectional planner only spends few cycles more. In fact the bidirectional planner "forces" the search around the newly assigned goal. This leads the evaluations, and hence the action probabilities, to change according to the new goal until enough confidence is gained.

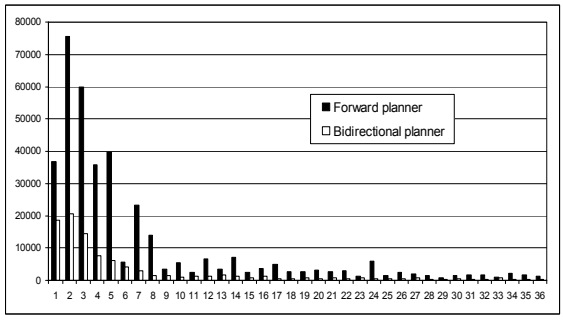

Figure 7: Cycles spent planning by the forward and bidirectional planners before achieving the goals, for the first 36 successes. Average over 24 seeds.

Limits of the Controllers

These encouraging results about the planning algorithms are weakened by some drawbacks. The forward planner and even more so the bidirectional planner have a quite complex architecture and are composed of heterogeneous neural networks. The functioning of both algorithms relies on the possibility of building a sufficiently reliable model of the world. Moreover, contrary to reinforcement learning, the planning algorithms need to explore the world before attempting to solve the tasks in order to acquire such model. Backward planning relies on the possibility of training the back-actor to "guess" what action could have led to a particular state: it is not clear if this is possible within problem domains different from navigation. In 8 seeds out of 32 the experts of the modular evaluator fail to specialise, while this specialisation seems necessary to solve the multi-goal task.

4. Summary and Conclusion

This paper has tested two neural planners presented previously, with a simulated robot engaged in a new multi-goal stochastic shortest-path problem. To cope with catastrophic interference, the critic, the actor and the back-actor components, previously implemented with monolithic neural networks, have been replaced with modular networks. The algorithm presented in [6] (suitable for the supervised training of "mixture of experts networks") has been adapted to train the critic, while an unsupervised learning algorithm has been used to train the actor's new hierarchical architecture.

The results of simulations have shown that the specialisation of the critic's experts (one per goal) and the partial specialisation of the actor's experts (one prevailing expert for goal) allows the system to cope with multi goals. These results suggest that emergent functional modularity is a useful mechanism to use to cope with interference both in the case of reinforcement learning and planning so to build scalable controllers.

Other results have confirmed the outcome of single-goal tests. The planners showed a superior efficiency vs. sheer reinforcement learning from the very first time a goal was assigned to the robot. With successive experiences the performance further improved, and the robot became "confident" enough about how to reach the different goals and did not need to plan anymore. Moreover the bidirectional planner outperformed the forward planner both in terms of action cycles and planning cycles needed to achieve goals, thanks to its capacity to focus exploration around the start and the goal and to propagate the evaluations quickly.

Notwithstanding these encouraging results, the controllers studied present some drawbacks as the need of an accurate (though not perfect) model of the world, and a high degree of complexity of the architecture.

The controllers proposed here should be applicable to real robots. In fact they are capable of dealing with noise both at the level of sensors and effectors thanks to the use of neural networks and to the use of "policies" for planning [2]. However two problems could arise in doing so. The first is that it may be difficult to train the predictor with success. Perhaps a solution would be to implement planning at a more abstract level where prediction is easier [2]. The second is that the pre-processing with contrasts is too simple to disambiguate the images of different positions. A solution would be to use a more sophisticated pre-processing (e.g. cf. [13]). The controllers should be also applicable to problem domains different from navigation (some problems may arise for the implementation of the back-actor, cf. [2]).

5. Acknowledgements

The University of Essex has funded the author's research. Special thanks are addressed to Prof. Jim Doran for the precious discussions of the ideas presented in the paper and James Adam for his kind help during the preparation of the paper.

6. References

[1] Baldassarre G., A Modular Neural-Network Model of the Basal Ganglia's Role in Learning and Selecting Motor Behaviours, in Altmann E.M., Cleermans A., Schunn C.D., Gray W.D. (eds.), *Proceedings of the Fourth International Conference on Cognitive Modelling*, pp. 37-42, Lawrence Erlbaum Associates, Mahwah NJ, 2001.

[2] Baldassarre G., *Planning with Reinforcement Learning and Neural Networks*, PhD thesis, Department of Computer Science, University of Essex, 2001. Submitted.

[3] Barto A.G., Bradtke S.J., Singh S.P., Learning to act using real-time dynamic programming, *Artificial Intelligence*, vol. 72, pp. 81-138, 1995.

[4] Calabretta R., Nolfi S., Parisi D., Wagner, G.P., Emergence of functional modularity in robots, in Pfeiffer R. (ed.), *From Animals to Animats 5: Proceedings of the 5th International Conference on the Simulation of Adaptive Behaviour,* pp. 497-504, The MIT Press, Cambridge MA, 1998.

[5] 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*, pp. 345-352, The MIT Press, Cambridge MA, 1995.

[6] Jacobs R.A., Jordan M.I., Nowlan S.J., Hinton G.E., Adaptive mixtures of local experts, *Neural Computation*, vol. 3, pp. 79-87, 1991.

[7] Korf R.E., Optimal path finding algorithms, in Kanal L.N., Kumar V. (eds.), *Search in Artificial Intelligence*, Springer-Verlag, Berlin, 1988.

[8] Lin L.J., Self-improving reactive agents based on reinforcement learning, planning and teaching, *Machine Learning*, vol. 8, pp. 293-391, 1992.

[9] Nolfi S., Elman J.L., Parisi D., Learning and evolution in neural networks, *Adaptive Behavior,* vol. 3, pp. 5-28, 1994.

[10] Ross S., *Introduction to Stochastic Dynamic Programming*, Academic Press, New York, 1983.

[11] Sharkey N.E., Sharkey A.J.C., An analysis of catastrophic interference, *Connection Science*, vol. 7, pp. 301-329, 1995.

[12] 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*, pp. 216-224, Morgan Kaufmann, San Mateo CA, 1990.

[13] Sutton R.S., Barto A.G., *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge MA, 1998.

[14] Thrun S., *Efficient Exploration in Reinforcement Learning*, Carnegie-Mellon University, Technical Report CMU-CS-92-102, 1992.

[15] Widrow B., Hoff M.E., *Adaptive Switching Circuits*, IRE WESCON Convention Record, Part IV, pp. 96-104, 1960.