2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

An Improved Minibrain That Learns Through Both Positive and Negative Feedback

Chee Wee Phua and Alan Blair

***Abstract*— A new reinforcement learned neural network, that follows the ideas of the minibrain network but includes exploration and learns through both positive and negative feedback, is proposed. The proposed ReL network is evaluated against the minibrain network in the $n \times n$ grid world domain and the taxi domain and is shown to perform significantly better than the minibrain network.**

I. INTRODUCTION

Neural networks were originally proposed to model the biological brain. Due to their inherent parallel processing and generalization ability, they have found their way into many interesting applications such as driving a vehicle on a highway [1] and face recognition [2].

However, most neural networks used today are trained through supervised learning. Although such a network will be able to generalize to achieve the task in the particular manner it has been taught, it will not be able to explore novel ways to achieve its goals.

Reinforcement learning is another training method that is capable of learning such novel goal directed behavior. It has been used successfully to train neural networks in a number of domains such as controlling an inverted pendulum [3], and landing an aircraft [4].

Previous work on reinforcement learned neural networks can be categorized into three classes: actor only networks, critic only networks, and actor critic networks. Actor only networks [5][6] learn the policy directly by adjusting their parameters in the direction of improvement as indicated by the reinforcement signal. Critic only networks [7] learn an approximation of the value function, which is then used to derive the policy. Actor critic networks combine the above networks by feeding the output of the critic network to the actor network.

Recently an actor only network, the minibrain network, has been proposed by Chialvo et. al. [8]. Their approach, which can be applied to almost any network architecture, is based on two principles: 1) a winner take all approach and 2) learning only upon punishment.

In this paper we propose a new reinforcement learned neural network, that follows the ideas of the minibrain network but includes exploration and learns through both positive and negative feedback.

This paper is organized as follows. Sections II and III review the relevant concepts in reinforcement learning, and previous work on minibrain networks respectively. Section IV introduces the new reinforcement learned neural network. Section V introduces the two domains which will be used to test and compare the performance of the reinforcement learned neural networks. Section VI presents and compares the results of the reinforcement learned neural networks in these domains. And finally, Section VII concludes the paper and provides directions for further work in the area.

Chee Wee Phua and Alan Blair are with National ICT Australia, University of NSW, Australia (email: cheewee.phua@nicta.com.au, blair@cse.unsw.edu.au).

II. REINFORCEMENT LEARNING

In reinforcement learning [9], an agent learns to perform a task through trial and error. This is characterized by the agent sensing its environment through sensors and acting on it through effectors. The goal of the agent is to learn a policy for choosing actions which maximize its cumulative reward. By giving the agent an appropriate performance measure signal – reward in the case of success or punishment in the case of failure – the agent should be able to learn the actions required to perform the task. Such a performance measure signal is often only available after the agent has performed a sequence of actions. This introduces the *credit assignment problem*: which components of that sequence are primarily responsible for the success or failure.

As the agent learns through trial and error, it has to explore unknown states and actions in order to learn. On the other hand, it must also exploit the states and actions which it has already learned will yield high reward. This creates the second problem in reinforcement learning, the problem of balancing *exploration* and *exploitation*. To solve this problem, an appropriate action selection strategy has to be used. There are three well known selection strategies:

1) The greedy selection strategy. With this strategy, the agent will always select the action which is expected to yield the highest cumulative reward.
2) The ε-greedy selection strategy. With this strategy, the agent will with a probability of ε, choose the action to take randomly with equal probability for each action.
3) The soft max selection strategy. With this strategy, the agent will take an action a in state s with a probability which is proportional to some strictly increasing function of a predetermined parameter.

This concludes the quick overview of reinforcement learning. The interested reader should refer to [9] for a more comprehensive discussion.

III. MINIBRAIN

One approach to solving the *credit assignment problem* is the minibrain network proposed by Chialvo et. al. [8]. These networks solve the credit assignment problem through the use of what the authors term as *extremal dynamics*. This is essentially a winner take all approach, in which firing activity only propagates through the strongest connections. This reduces the number of nodes active at any one time and thus simplifies the credit assignment problem. After every activation, the activated connections are then 'tagged' for punishment or reward upon failing or succeeding in the task.

Another characteristic of the minibrain network is that it only learns upon punishment: the weights are adjusted only when the reward is negative. This is claimed to increase the speed of learning in the minibrain network. Upon receiving a negative reinforcement value, regardless of its magnitude, the weights of the connections which were previously active will be reduced by a constant or random amount.

A modified version of the original minibrain network which Klemm et. al. [10] used to solve the XOR problem will now be presented. In this work [10], the authors selected the node to activate stochastically where the probability of firing for each node j is:

$$p_j = a^{-1} e^{\beta h_j} \tag{1}$$

where $h_j = \sum_i w_{ij} x_i$ is the sum of weighted input for node j, $a = \sum_j e^{\beta h_j}$ is the normalization factor, and β is the Boltzmann noise factor.

Note that when $\beta \to \infty$, this becomes the greedy selection strategy, where the node with the maximum sum of weighted inputs is always selected, and when β is finite it becomes a version of the soft max selection strategy.

Another extension made in their work is the introduction of selective punishment. This is done by giving connections which were previously successful a number of *chances* before their weights are reduced. This is defined by the following equations:

$$c_{ij}(t+1) = \begin{cases} \Theta, & \text{if } c_{ij}(t) + r > \Theta \\ c_{ij}(t) + r, & \text{if } \Theta \geq c_{ij}(t) + r \geq 0 \\ 0, & \text{if } 0 > c_{ij}(t) + r \end{cases}$$

$$w_{ij}(t+1) = \begin{cases} w_{ij}(t) - \delta, & \text{if } c_{ij} = 0 \\ w_{ij}(t), & \text{otherwise} \end{cases} \tag{2}$$

where $c_{ij}(t)$ is the chance value for the connection between nodes i and j at time t, Θ is the maximum value for the chance values, r is the reinforcement signal supplied, positive for reward and negative for punishment, $w_{ij}(t)$ is the weight of the connection between nodes i and j at time t, and δ is a constant value.

The activity propagation and weights updating procedure used by Klemm et. al. [10] are summarized in table I.

Bak et. el. [11] also incorporated a selective punishment approach into the original minibrain network and used it to learn a simple sequence following task. In this work however, rather than giving previously successful connections chances, the weights of these connections are reduced by a smaller amount when a punishment is received.

TABLE I
THE ACTIVITY PROPAGATION AND WEIGHTS UPDATING PROCEDURE USED BY KLEMM ET. AL. [10].

```
Activate(Input = $X$)
  1   Activate input layer according to $X$
  2   From the first to the last layer do
  3      Select the node $j$ to activate where the probability of
           selecting each node is defined in equation 1
  4      Activate node $j$
  5   Mark all activated connections

Update(Reward = $r$)
  1   For each connection in the network do
  2      If connection is activated then
  3         update weight of connection according to equations 2
  4   Unmark all connections
```

IV. ReL Network

In this section, the **Re**inforcement **L**earned (ReL) network will be introduced. The ReL network model is quite similar to the minibrain network model in that:

- activity in the network only propagates through the strongest connections,
- all activated connections are marked and updated only when a reinforcement signal is received, and
- each node has only two states: activated, having a value of 1, and unactivated, having a value of 0.

However, it differs from the minibrain network model in that:

- a hybrid of ε-greedy and soft-max selection strategy is used to select the node to activate, and
- the weight modifications depend on the size of rewards and the time the reward is received.

The differences in activity propagation and weight updating will be detailed in the following subsections.

A. Activity Propagation

The exploration strategy used in the ReL network is a hybrid of the ε-greedy and soft-max selection strategy. The reasons for the hybrid approach are:

1) Although the ε greedy selection strategy will randomly choose a node to activate with a probability of ε, it chooses such a node with equal probability for all nodes. This is not ideal in that it will choose between a good node and a bad one with equal probability.
2) Although the soft-max selection strategy will select a node to activate with a probability proportional to its sum of weighted inputs, it does so all the time. This is not ideal in that it might cause too much exploration.

Hence to select the node k which will be activated with a probability proportional to its sum of weighted inputs, the following equation will be used:

$$k = \max_j \left(\sum_i (w_{ij}(t) + \nu_{ij}(t)) x_i \right) \tag{3}$$

where $w_{ij}(t)$ is the weight for the connection between nodes i and j at time t, x_i is the value of node i, and $\nu_{ij}(t)$ is the noise for the connection between nodes i and j at time t.

TABLE II

THE ACTIVITY PROPAGATION PROCEDURE OF THE ReL NETWORK.

```
Activate(Input = $X$)
 1    $t = t + 1$
 2    Activate input layer according to $X$
 3    From the first to the last layer do
 4        For each connection connecting to the previous layer do
 5            Determine the $\nu_{ij}$ value according to equation 4
 6        Determine node $k$ according to equation 3
 7        Activate node $k$
 8        For each connection connecting to the previous layer do
 9            determine the $a_{ij}$ value according to equation 5
10        Mark all activated connections
```

The rate of exploration is then controlled by controlling the rate of change of ν_{ij} with the following rule:

$$\nu_{ij}(t) = \begin{cases} \text{random()} & \text{if node } i \text{ is activated and with a} \\ & \text{probability of } \varepsilon \\ \nu_{ij}(t-1) & \text{otherwise.} \end{cases} \tag{4}$$

where ε is the probability of taking a random action, and random() is a random number generator with an expected value of 0.

Note that if $\varepsilon = 1$, the selection strategy becomes a pure soft-max selection strategy, and if $\varepsilon = 0$ the selection strategy becomes the one used in the greedy minibrain network.

Then as in minibrain, activated connections are marked and will be updated when a reinforcement signal is received. But instead of marking with just a boolean mechanism, the time of activation is also recorded. This is done by letting:

$$a_{ij}(t) = \begin{cases} t & \text{if connection between } i \text{ and } j \text{ is} \\ & \text{activated} \\ a_{ij}(t-1) & \text{otherwise.} \end{cases} \tag{5}$$

Thus $a_{ij}(t)$ is the last activation time (equal or prior to t) for the connection between nodes i and j. We explain later how $a_{ij}(t)$ will be used as a factor in the updating of the weights. Table II gives a summary of the activity propagation procedure.

Example 1: To illustrate this procedure, we will use a single layered feed forward network as shown in figure 1, and present the network with the input $< 1, 1, 0 >$. Assuming that the network had received three inputs previously and has yet to receive any reinforcement signal; the current value of t is 3 and the current w_{ij}, ν_{ij}, and a_{ij} values for the network are as shown in figure 2.

In the first step of the activation procedure, the time step is incremented, i.e. $t = t + 1 = 4$. In step 2, we activate nodes 1 and 2. Then in step 3, we enter a loop to propagate the activity through the layers. Next in steps 4 to 5, we determine the ν_{ij} values for the connections between the input layer and layer one using equation 4. Since nodes 1 and 2 are the only presynaptic nodes that are activated, we will only assign random noise to $\nu_{14}, \nu_{15}, \nu_{24}$, and ν_{25} with a probability of ε. Let us assume that ν_{14} and ν_{25} are assigned random noise values of -0.039 and 0.074 respectively. Then

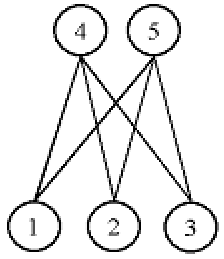

Fig. 1. This figure shows the one layered neural network used in the examples. It has 3 input nodes (nodes 1, 2 and 3) and 2 output nodes (nodes 4 and 5).

$t = 3$

w	4	5
1	0.345	-0.124
2	0.502	0.414
3	-0.349	0.246

ν	4	5
1	0.004	0.188
2	-0.273	0
3	0	-0.016

a	4	5
1	2	1
2	2	0
3	0	3

$t = 4$

w	4	5
1	0.345	-0.124
2	0.502	0.414
3	-0.349	0.246

ν	4	5
1	-0.039	0.188
2	-0.273	0.074
3	0	-0.016

a	4	5
1	2	4
2	2	4
3	0	3

Fig. 2. This figure shows the initial ($t = 3$) and resulting ($t = 4$) w_{ij}, ν_{ij}, and a_{ij} values of the network used in the activation procedure example.

in the step 6, we use equation 3 to determine the node to activate. According to the equation, the sum of weighted inputs for nodes 4 and 5 are 0.508 and 0.552 respectively. As node 5 is the node with the largest sum of weighted inputs, it will be the node that is activated in step 7. Then in steps 8 to 9, we will determine the a_{ij} values for connections from the input layer and layer one using equation 5. Since the connections from nodes 1 to 5, and 2 to 5 are the only ones that are activated, we set $a_{15} = a_{25} = 4$. Finally, we mark the connections connecting nodes 1 to 5, and 2 to 5 activated. Since we have only one layer in the network we exit the procedure. The resulting w_{ij}, ν_{ij}, and a_{ij} values for the network are shown in table 2.

B. Weight Updating

The weights in the ReL network are updated using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\gamma^{(t-a_{ij})} r \nu_{ij}(t) \tag{6}$$

where η is the learning rate, γ is the decay rate, and r is the reward received, ν_{ij} is the noise of the connection (equation 4), and a_{ij} is the time of activation (equation 5).

The justification for this learning rule is as follows. Modifying the weights in proportion to the received reward value increases the probability of activation for connections whose activation leads to a high reward and conversely reduces the probability of activation for those whose action leads to a low reward.

The equation apportions more responsibility to the connections whose last time of activation is closer to the time the reward is received. This is achieved by the $\gamma^{(t-a_{ij})}$

TABLE III

THE WEIGHT UPDATING PROCEDURE OF THE REL NETWORK.

```
Update(Reward = $r$)
1   $t = t + 1$
2   For each connection in the network do
3     If connection is activated then
4       update weight of connection according to equation 6
5   Unmark all connections
6   Set all $a_{ij}$ and $\nu_{ij}$ values to 0
```

$t = 5$

w	4	5
1	0.34149	-0.10708
2	0.47743	0.42066
3	-0.349	0.24456

ν	4	5
1	0	0
2	0	0
3	0	0

a	4	5
1	0	0
2	0	0
3	0	0

Fig. 3. This figure shows the resulting ($t = 5$) w_{ij}, ν_{ij}, and a_{ij} values of the network used in the weight updating procedure example.

term which causes greater adjustment of weights to those connections.

The noise of the connection used in equation 3 serves as a point of convergence. The idea is that each weight should be changed such that it is moving towards the value of the noise that was added which led to its activation. Since the expected value of the noise is 0, the weight of the connection will eventually fluctuate around some point when there is constant activation and reward.

Finally, as in any neural network learning algorithm, the changes are made in small steps, controlled by the learning rate, so as to converge.

After updating the weights, all connections will be unmarked, and all a_{ij} and ν_{ij} values will be reset to 0. Table III gives a summary of the weight updating procedure.

Example 2: To demonstrate the weight updating procedure, let us use the resulting w_{ij}, ν_{ij}, and a_{ij} values of the example from the previous section. In this example, the values of the parameters used in the weight updating procedure will be $\eta = 0.1, \gamma = 0.9$ and $r = 1$.

In the first step of the procedure, the time step is incremented, i.e. $t = t + 1 = 5$. In steps 2 to 4, we update the weights of the connections that were previously activated. Hence all weights in the network except w_{34} are candidates for update. As an example,

$$w_{35} = 0.246 + 0.1 \cdot 1 \cdot 0.9^{(4-3)} \cdot -0.016 = 0.24456$$

Then in step 5, all connections are unmarked. Finally in step 6, all a_{ij} and ν_{ij} values will be set to 0. The post update w_{ij}, ν_{ij}, and a_{ij} values of the network are shown in figure 3.

V. EXPERIMENTS

This section will describe two domains, the $n \times n$ grid world domain and the taxi domain, which will be used to test and compare the performance of the reinforcement learned neural networks

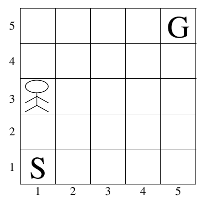

Fig. 4. This figure shows a 5×5 grid world. The **S**tarting position is at the lower left corner of the grid world, the **G**oal is at the upper right corner, and the player is at position $< 1, 3 >$.

A. The $n \times n$ Grid World Domain

The simple grid world game has often been used in the area of reinforcement learning ([9], [12], and [13]) to test and illustrate various algorithms.

In the $n \times n$ grid world game, the objective of the game is to start in the lower left corner of the grid world and navigate to the goal in the upper right corner of the grid world using the minimal number of steps. Figure 4 shows an example of a 5×5 grid world.

At each position, the player can either move *Left, Right, Up,* or *Down* which will take it to the respective adjacent square. A move into the wall leaves the player's position unchanged. As an example: in figure 4, a *Left* action will leave the player position unchanged at $< 1, 3 >$, a *Right* action will take the player to position $< 2, 3 >$, an *Up* action will take the player to position $< 1, 4 >$, and a *Down* action will take the player to position $< 1, 2 >$.

When the player reaches the goal it will be rewarded with a reinforcement value of 1 and the game will be restarted. As the player might wander around in the grid world forever without reaching the goal position, a cap will be placed on the number of steps which can be taken before the game is terminated. If the player fails to reach the goal within 1000 steps, it will be punished with a reinforcement value of -1 and the game will be restarted.

To represent the state of the game, 2 groups of n input nodes will be used. Each group will be mapped one to one to each axis of the grid world and each node of each group will be mapped one to one to a position in that axis. If the player is in the position the input node represents, the input node will be activated by giving it a value of 1, otherwise it will be given a value of 0.

To represent the actions of the game, 4 output nodes will be used. Each node will be mapped one to one to each of the 4 actions. If an output node is activated, i.e. having an output value of 1, then the player will take the action represented by the output node.

As an example, figure 5 shows the representation of the state shown in figure 4 and taking a *Left* action in that state.

B. The Taxi Domain

The taxi game is used in [14] and [15] to demonstrate and evaluate hierarchical reinforcement learning algorithms.

The objective of this game is to navigate a taxi around a 5×5 grid world to pick up a passenger at the pickup

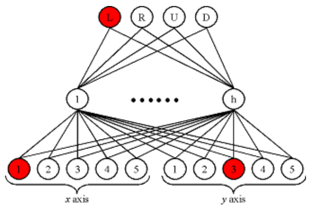

Fig. 5. This figure shows the network representation of taking a *Left* action in the state shown in figure 4. Activated nodes are highlighted.

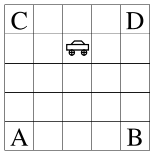

Fig. 6. This figure shows the pickup and drop off locations (marked A, B, C, and D) in the taxi game.

location and drop the passenger off at the drop off location. There are 4 possible pickup and drop off locations, which are the corners of the grid world as shown in figure 6.

At the start of every game, the taxi, pickup and drop off locations will be selected randomly. Then as in the grid world game, the taxi can either move *Left, Right, Up,* or *Down*, each with the same consequence as in the grid world game. However, unlike the grid world game, 2 more actions will be available for each position in the game:

- *Pick Up* - Picks the passenger up if the taxi is at the pickup location and the passenger is not picked up yet. Otherwise the current state is left unchanged.
- *Drop Off* - Drops the passenger off if the taxi is at the destination location and the passenger is in the taxi. Otherwise the current state is left unchanged.

As an example, executing the *Pick Up* action in the left of figure 7 will pick the passenger up, but executing the *Drop Off* action will not affect the state. However, executing the *Pick Up* action in the right figure will not affect the state, but executing the *Drop Off* action will drop the passenger off, and thus complete the task.

When the passenger is dropped off at the drop off location the taxi will be rewarded with a reinforcement value of 1 and

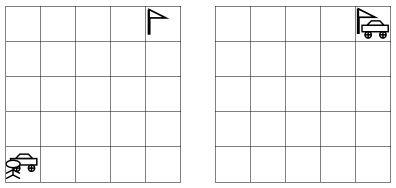

Fig. 7. In the left figure, the passenger has not yet been picked up and is waiting at the lower left corner of the grid world. Since the taxi is also in that position, executing the *Pick Up* action will pick the passenger up. In the right figure, the passenger had been picked up and the taxi is in the drop off location (at the upper right corner of the grid world), hence executing the *Drop Off* action will drop the passenger off, and thus complete the task.

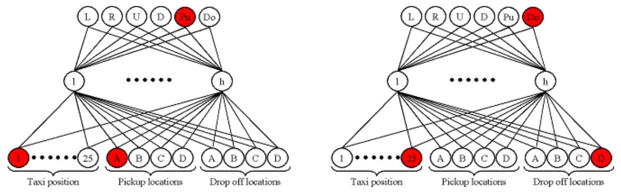

Fig. 8. The left figure shows the network representation of taking a *Pick up* action in the state shown in the left of figure 7. The right figure shows the network representation of taking a *Drop off* action in the state shown in the right of figure 7

the game will be restarted. As with in the grid world, the taxi might wander around and never complete the task, therefore a cap will be placed on the number of steps which can be taken before the game is terminated. If the taxi fails to drop the passenger off at the drop off location within 1000 steps, it will be punished with a reinforcement value of -1 and the game will be restarted.

To represent the state of the game, 25+4+4 input nodes will be used. 25 of the input nodes will represent the taxi location. Each of these 25 will be mapped one to one to a position in the 5×5 grid world. Then only the input neuron that represents the position the taxi is in will be activated (by giving it a value of 1), and the other 24 will be inactivated (by giving them a value of 0).

The remaining input nodes will then be divided into 2 groups of 4, with one group representing the pickup locations, and the other representing the drop off locations. Each neuron in each group will be mapped one to one to a pickup or drop off location in the game (i.e. a corner of the grid world). If the passenger is awaiting pickup at one of the pickup locations, then only the input neuron representing that pickup location will be activated, and the remaining 7 (the other 3 representing the pickup locations, together with all of the input nodes representing drop off locations) will be inactivated. Similarly, if the passenger has been picked up, then only the input neuron representing the drop off location will be activated, and the remaining 7 (the other 3 representing the drop off locations, together with all of the input nodes representing pick up locations) will be inactivated.

To represent the actions of the game, 6 output nodes will be used. Each node will again be mapped one to one to each of the 6 actions. If an output neuron is activated, i.e. having an output value of 1, then the taxi will take the action represented by that output neuron.

As an example, the left of figure 8 shows the representation of the state shown in the left of figure 7 and taking a *Pick up* action in that state. The right of figure 8 shows the representation of the state shown in the right of figure 7 and taking a *Drop off* action in that state.

C. Experimental Setup

The experiments will employ two versions of the minibrain network used by Klemm et. al [10] (table I): 1) an exploratory version by setting $\beta = 10$, and 2) a greedy version by setting

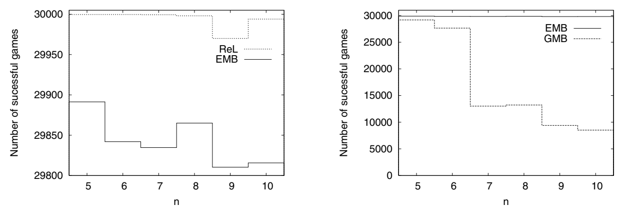

Fig. 9. These figures show the number of successful games, i.e. those in which the goal is reached, achieved by the various reinforcement learned neural networks in the $n \times n$ grid world as n increases. Left: ReL versus exploratory minibrain (EMB). Right: EMB versus greedy minibrain (GMB).

$\beta = \infty$. In both versions, the other parameters used will be the same as the ones used by Klemm et. al. [10], that is, $\delta = 1$, and $\Theta = 2$.

For the parameters in the ReL network, we use the values $\varepsilon = 0.1, \gamma = 0.999$, and $\eta = 0.2$. The random number generator used will have a range of $[-0.5, 0.5]$ with uniform distribution.

The networks used in the experiments will be two layer feed forward networks each with 300 hidden nodes. The results presented, unless otherwise stated, will be the average of 3 runs with each run consisting of 30 000 games.

VI. Results

This section will describe and discuss the results of the experiments conducted with the ReL network and the minibrain network.

A. Results in $n \times n$ Grid World

Now the results from the $n \times n$ grid world game will be presented. We will start with the the number of successful games – those in which the goal is achieved by the network – as n increases (figure 9). This is a good indicator of the learning speed of the network, since the faster a network learns, the greater number of successful games it will achieve (within the total of 30 000 games).

The exploratory minibrain seems to learn the task quite quickly, regardless of the size of the grid world. But as the size of the grid world increases, the number of successful games achieved by the greedy minibrain decreases, thus indicating a decrease in the learning speed.

The reason for the slow learning is lack of exploration. Without exploration, the greedy minibrain will always take the same action in the same state until it is punished when the 1000 step limit is reached. This is because the minibrain will learn, in other words change its policy, only when it is punished. Thus, in order for the network to reach the goal, it will pursue a more or less random search in policy space in the hope of stumbling upon a successful policy. The chance of finding a successful policy with such an unstructured search decreases dramatically as the size of the grid world increases.

The ReL performs better than both versions of the minibrain network in this aspect. As with the case of exploratory versus greedy minibrain, the inclusion of exploration in the ReL network allows it to maintain its performance as the size of the grid world increases. Moreover with learning on both reward and punishment, and being able to distinguish good solutions from poor ones, the ReL network is able to learn and benefit from the experience it has gained from exploration, and achieve a much better performance than the exploratory minibrain.

Next we will look at the number of steps taken by the networks during an average game (figure 10). For the exploratory minibrain, the number of steps taken is random but largely under 600. The reason for this is that the minibrain network only learns when it is punished. This has two consequences which explain the random result:

1) As the network is only punished when the number of steps taken exceeds 1000, the network is seldom punished and thus rarely learns.
2) The experiences gained from a successful exploration are not learned; a solution which takes the optimal number of steps (10, for the 6×6 world) and one that takes 900 steps are the same to the minibrain network. It is unable to differentiate between a poor solution and a good one.

For the greedy minibrain, the number of steps taken to reach the goal decreases in a stepwise manner as a result of averaging over the three runs. The reason behind this is the lack of exploration. Once the greedy minibrain starts to find a solution, it will start exploiting it without looking for any alternate solution. In this case, two of the three runs have found an optimal solution while the third run has gotten stuck in a suboptimal solution. Thus the (cumulative) percentage of optimal runs averages out to around 60%, as shown in the bottom of figure 10.

Again, the ReL network, unlike the exploratory minibrain, is able to learn a reliable way to reach the goal. Also as a result of exploration the ReL network starts to learn much faster than the greedy minibrain network. However, due to continued exploration, the number of steps taken is variable unlike the greedy minibrain. Despite this, the percentage of games in which it reaches the goal in an optimal way is higher for the ReL network than for either the exploratory minibrain or the greedy minibrain. This is once again attributed to the exploration, learning on both reward and punishment, and being able to distinguish good solutions from poor ones.

B. Results in the Taxi Domain

We will now turn to the results of the networks in the Taxi domain. Figure 11 shows the number of successful games, games in which the passenger is delivered successfully, achieved by each network. The networks with exploration (the exploratory minibrain network and the ReL network) manage to deliver the passenger successfully for a large portion of the games. However, the greedy minibrain network fails to learn this task. This comes as no surprise since the action state space of the taxi game ($25 \times 4 \times 2 \times 6$) is much larger that that of a 10×10 grid world game ($10^2 \times 4$).

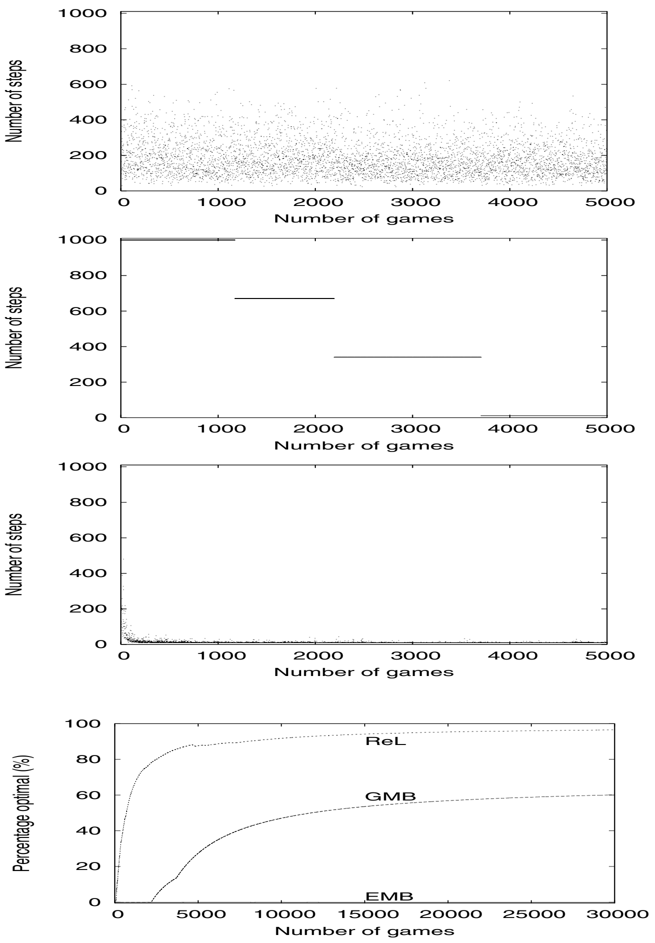

Fig. 10. These figures show a comparison on the number of steps taken by the networks to reach the goal in a 6 $\times$ 6 grid world, averaged over three runs. The first three figures show the number of steps taken to reach the goal by the exploratory minibrain (EMB) network (top), the greedy minibrain (GMB) network (middle), and the ReL network (lower) for the first 5k episodes. In episodes 5k to 30k, the distribution of results remains similar to that between 4k and 5k; however the (cumulative) percentage of games in which the networks reached the goal in an optimal way continues to grow, as shown in the bottom figure.

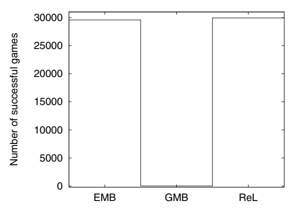

Fig. 11. Comparison of results in the taxi domain. This figure shows the number of successful games, games in which the the passenger is delivered correctly, achieved by the exploratory minibrain network(EMB), the greedy minibrain network (GMB), and the ReL network.

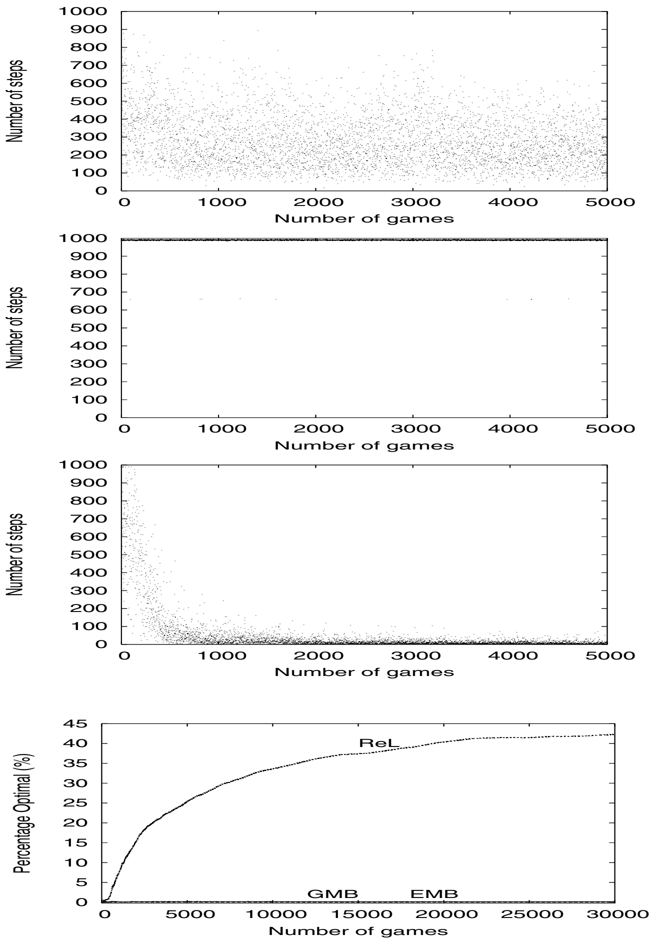

Fig. 12. Comparison on the number of steps taken by the networks to complete the task in the taxi domain. The first three figures show the number of steps taken to complete the task, by the exploratory minibrain (EMB) network (top), the greedy minibrain (GMB) network (middle), and the ReL network (lower) for the first 5k episodes. The bottom figure shows the (cumulative) percentage of games in which the networks completed the task in an optimal way, for the entire run (games 0 to 30k).

Therefore, the greedy minibrain will have greater difficulty finding a policy enabling it to complete the task during its punishments.

Moving on to the number of steps taken by the networks during the games (figure 12), as the greedy minibrain fails to learn in the game, the number of steps it takes stays largely at the 1000 step mark (although it occasionally drops to a lower level indicating successful delivery of the passanger in certain rare circumstances). For the same reasons identified in section VI-A, the number of steps taken by the exploratory minibrain is random but largely under 700 steps. Finally for the ReL network, the number of steps declines steadily at the start before stabilizing below the 100 step mark. The sporadic spikes in the number of steps taken is again a result of constant exploration.

Next, we will look at the percentage of games in which the networks complete the task in an optimal way (bottom of figure 12). Both versions of the minibrain fail to complete the task in an optimal way, since the greedy minibrain fails to learn and the exploratory minibrain produces unreliable results. But for the ReL network, it manages to complete the task optimally around 42% of the time. This relatively low figure is a result of over exploration - as can be seen in the left of figure 13, where the percentage of optimal task completion increases as the ε value, the value which controls

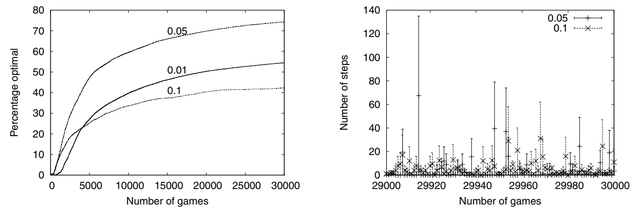

Fig. 13. The left figure shows the (cumulative) percentage of games in which the ReL network delivers the passenger in an optimal way for different ε values. The right figure shows the range of the number of steps taken by the ReL network during the last hundred games when $\varepsilon = 0.05$ and when $\varepsilon = 0.1$. A lower ε value results in relatively infrequent but longer spikes, while a higher value results in more frequent short spikes.

the rate of exploration in the ReL network, is reduced from 0.1 to 0.05. However if ε is reduced further to 0.01, the percentage of optimal completions decreases again, this time due to under exploration.

Decreasing the ε value however is a double edged sword; although it does increase the percentage of optimal task completion, it also increases the number of steps taken when the ReL network does complete the task sub-optimally (see right of figure 13). This is a result of the persistence of the noise value used in the activity propagation procedure (table II). To better illustrate this, let us consider an example of selecting between 2 nodes, A and B, where node A has a higher weight value and is the node which will lead to optimal task completion. Suppose during one execution of activity propagation procedure, the noise values assigned to the nodes by equation 4 are such that the sum of weighted inputs (defined by equation 3) of node B is greater than that of A. This results in node B being activated, and thus the suboptimal action being taken. This will persist until a subsequent execution of the procedure, where the noise values assigned by equation 4 are such that the sum of weighted inputs of node A is greater than that of node B. Therefore as we decrease the value of ε, the value which controls the rate of exploration, not only do we decrease the probability of selecting a suboptimal node, we also decrease the chance of restoring the optimal node, resulting in the relatively infrequent but longer spikes when $\varepsilon = 0.05$ compared to when $\varepsilon = 0.1$.

VII. Conclusion

In this paper, we proposed a new reinforcement learned neural network algorithm, that follows the ideas of the minibrain network but includes exploration and learns through both positive and negative feedback. The proposed ReL network is evaluated against the minibrain network in the $n \times n$ grid world domain and the taxi domain and is shown to perform significantly better than the minibrain network. Specifically:

- The greedy minibrain network fails to learn the tasks.
- The exploratory minibrain network learns to complete the tasks, but suboptimally and unreliably.
- The ReL network learns to complete the tasks reliably and, most of the time, optimally.

In future work, we plan to apply the ReL network to a greater variety of tasks, and to explore possible connections with synfire chains [16] and other learning algorithms.

References

[1] T. Jochem, D. Pomerleau, and C. Thorpe, "Vision-based neural network road and intersection detection and traversal," in *IEEE Conference on Intelligent Robots and Systems (IROS '95)*, vol. 3, August 1995, pp. 344 – 349.

[2] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," in *Computer Vision and Pattern Recognition '96*, June 1996.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, pp. 835–846, 1983.

[4] D. V. Prokhorov, R. A. Santiago, and D. C. W. II, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, 1995.

[5] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[6] P. Marbach and J. Tsitsiklis, "Simulation-based optimization of markov reward processes," 1998. [Online]. Available: citeseer.ist.psu.edu/marbach98simulationbased.html

[7] D. V. Prokhorov and D. C. Wunsch II, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, September 1997. [Online]. Available: citeseer.ist.psu.edu/prokhorov97adaptive.html

[8] D. R. Chialvo and P. Bak, "Learning from mistakes," *Neuroscience*, vol. 90, no. 4, pp. 1137–1148, 1999.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[10] K. Klemm, S. Bornholdt, and H. G. Schuster, "Beyond hebb: Exclusive-or and biological learning," *Physical Review Letters*, vol. 84, no. 13, pp. 3013–3016, March 2000.

[11] P. Bak and D. R. Chialvo, "Adaptive learning by extremal dynamics and negative feedback," *Phys. Rev. E*, vol. 63, no. 3, p. 12, March 2001.

[12] J. Hu and M. P. Wellman, "Nash q-learning for general sum stochastic games," *Journal of Machine Learning Research*, vol. 4, pp. 1039–1069, 2003.

[13] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Eleventh International Conference on Machine Learning*. New Burnswick, 1994, pp. 157–163.

[14] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value functional decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[15] B. Hengst, "Discovering hierarchy in reinforcement learning with hexq," in *Proceedings of the Nineteenth International Conference on Machine Learning*. Morgan Kaufmann, 2002, pp. 243–250.

[16] M. Abeles, *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, 1991.