Timing is important: delaying action execution in Plastic Neural Networks

B. Torben-Nielsen G. de Croon E.O. Postma

Universiteit Maastricht
Institute for Knowledge and Agent Technologies (IKAT)
P.O. Box 616, 6200 MD Maastricht, the Netherlands
{b.torben-nielsen,g.decroon,postma}@cs.unimaas.nl

Abstract

Plastic Neural Networks (PNNs) are known for their ability to adapt to environmental changes. It is generally believed that PNNs cannot solve *timing tasks* which require a predefined delay before execution of an action. In this study we investigate the ability of PNNs to solve timing tasks. Our experiments evolve PNNs to perform successfully on a task requiring the delayed execution of an action. The results of our experiments show that PNNs are capable of solving the timing task. We analyse the underlying mechanism and find it is based on slow neural activation dynamics. The mechanism is discussed in relation to mechanisms found in other neural models. We conclude that any neural model that can accommodate slow activation dynamics can solve the timing task.

1 Introduction

Evolutionary robotics is a popular paradigm to develop neural controllers for robots [6]. Robotic systems built according to this paradigm have the property of adapting to dynamical environments without human intervention [4]. A technique which is especially articulated as adaptive and robust to environmental changes is the plastic controller based on Plastic Neural Networks (PNN) [4, 5, 12]. This technique is a neural networks with adaptive synapses which are updated on-line using simple learning rules. Plastic controllers proved to cope with environmental changes like changes in sensor response, changes in sensory appearance, rearrangement of environmental configuration, and, transfer across different robot platforms [4].

The adaptive capabilities of PNNs as compared to other neural models (such as the continuous time recurrent network - CTRNN), has been subject of debate. The debate started with a comparison between the CTRNN and PNN [2]. The light-switching task involves a rectangular arena with two locations: the home location and light-switch location (indicated by a black line on the wall of the arena). Starting from an initial position in between the two locations, the robot has to move towards the light switch and go further to the home location where is has to stay. The movement of the robot was restricted to forward and backward movement. Both the CTRNN and the PNN successfully performed the light-switching task. It was concluded that the PNN and CTRNN "displayed similar performances"[2].

As a reaction to these newly gained insights, a second comparison was conducted by [10]. They proposed that the similar results of the CTRNN and PNN found earlier were due to the ceiling effect: the task was relatively simple and both PNN and CTRNN performed optimal. To overcome the ceiling effect, a more complex task was employed in which the robot has to notice *in time* that another action was required for execution. In this timing task it was found that the CTRNN succeeded in 70% procent of the experimental runs and the PNN didn't succeed a single run. They concluded that "there may actually be significant differences in the ability of CTRNNs and PNNs to perform tasks requiring learning"; they attribute this difference to the inability of PNNs to delay the action execution.

In this study we want to clarify whether PNNs can time the action execution. The research question central to this stud reads: *can PNNs solve the timing task?*. An elementary timing task is used to investigate the ability to time actions. The experimental setup is described in the next section. The results from the experiment and an analysis of these results is given in Section 3. Finally, in Section 4 we discuss our results and provide a conclusion.

2 Experimental setup

The experimental setup involves the agent controller (subsection 2.1), the plastic neural network model (subsection 2.2), the genetic algorithm (subsection 2.3), and the experimental environment and task (subsection 2.4).

2.1 Agent controller

The robot is controlled by a plastic neural network with a fixed topology. As in [10] our network consists of 5 neurons: a bias neuron with constant input of 1, two input neuron activated by the distance sensors attached to the left and right side of the robot, respectively[1], one inter neuron or hidden neuron, and one output neuron whose output is linearly mapped onto the interval $]-10, 10[$ which is required for the simulator.

2.2 Plastic neural network model

Our PNN model is a fully recurrent plastic neural network and is based on [12]. The activity of neuron y_i is updated every sensory-motor cycle using the equation below.

$$y_i^t = \sigma(\sum\nolimits_{j=0}^{N} w_{ij}^{t-1} y_j^{t-1}) + I_i^{t-1} \tag{1}$$

[1]Since the robot is only permitted to move in one direction, the information picked up by the two sensors is redundant.

In the above equation, σ is the transfer function of the neuron and is defined as: $\sigma(x) = (1 + e^{x})^{-1}$. The sensory input to the i-th neuron, I_i is normalized onto the unit interval ($0 < I_i < 1$). Synapses are randomly initialised in the range $[0, 0.1]$ and updated every sensory-motor cycle according to the formula below.

$$w_{ij}^{t} = w_{ij}^{t-1} + \eta \Delta w_{ij} \tag{2}$$

With η representing the learning rate that can assume one of four discrete values, i.e., $\eta \in \{0, 0.3, 0.6, 0.9\}$. For the synaptic weight update Δw_{ij}, one of the four learning rules applies:

1. Plain Hebb rule: $\Delta w_{ij} = (1 - w_{ij}) x_i y_j$,
2. Post synaptic rule: $\Delta w_{ij} = w_{ij}(-1 + x_i) y_j + (1 - w_{ij}) x_i y_j$,
3. Presynaptic rule: $\Delta w_{ij} = w_{ij} x_i (-1 + y_j) + (1 - w_{ij}) x_i y_j$,
4. Covariance rule: $\Delta w_{ij} = \begin{cases} (1 - w_{ij}) \mathcal{F}(x_i, y_j) & \mathcal{F}(x_i, y_j) > 0 \\ w_{ij} \mathcal{F}(x_i, y_j) & otherwise \end{cases}$.

In the above equations x_i refers to the pre-synaptic neuron and y_j to the post-synaptic neuron of the connection with weight w_{ij}. The function $\mathcal{F} = tanh(4(1 - |x_i - y_j|) - 2)$ maps the difference between the activations x_i and y_j onto the interval $[-1, +1]$. All four learning rules employ a self-limiting component $(1 - w_{ij})$ to ensure that the synaptic strength remains on the unit interval and that the sign of the synapse is not changing.

2.3 Genetic algorithm (GA)

A genetic algorithm is used to optimise the parameters of the robot for the timing task. We employ synapse encoding so every synapse is represented by one gene. Each gene represents the three properties defining the behaviour of a synapse: sign, learning rate, and learning rule [4]. The population consists of 50 individuals. Initially, at $t = 0$ all individuals have random synaptic strengths. Individuals are tested in trials of 350 time steps and the 10 best individuals of the generation are used for reproduction. Reproduction is done with single point cross-over and gene mutation; cross-over probability was 0.3 and mutation occurred with probability 0.05.

2.4 Experimental environment and task

Figure 1 (left) illustrates the timing task. The agent is placed inside a tunnel and is able to move forwards and backwards. The target cannot be sensed by the robot and is located between the two parallel lines in front of the agent (only shown for illustrative reasons). The robot cannot perceive whether it is located in the target region or at some other position outside the tunnel, because the sensors are not activated at these locations. Successful performance in this task implies that the agent learns to delay the execution of actions. The robot is always starting facing forward and at approximately the same starting position as shown in Figure 1 (left). Individuals are evaluated using the following global external fitness function. The fitness is continuous, i.e., the fitness is adjusted after each performed step of the trial.

$$F = \sum_{t=1}^{t=\#steps} f_t, \text{ where } f_t = \begin{cases} 2 & \text{on target at step t} \\ -1.5 & \text{past target at step t} \\ \alpha = [0,1] & \text{elsewhere at step t} \end{cases}$$

Where α is defined as the normalized inverse distance to the target position. Individuals are encouraged to move forward and are rewarded when located in the target position. Penalties are given whenever an individual moves beyond the target location. The rationale behind the fitness function is that individuals should move as fast as possible towards the target position while decreasing their speed to null (until the target position is reached). A priori, we consider the possible evolution of three strategies for performing successful behaviour in the timing task: (1) continuous decrease of velocity until a complete standstill is reached at the target position, (2) driving forward until the end of the tunnel and then start decelerating as in (1), and, (3) driving forward out of the tunnel and start driving at a certain speed which maximizes the time is spent in the target position. The definition of the fitness function does not favour one of these strategies. Only the first 2 strategies show the capability of the controller to delay its action: the robot comes to a full stop when the neural activity of the motor neurons decays to null[2].

Experiments described in this report are performed in a modified version of the WSU Khepera simulator[9]. The Khepera robot and its sensory configuration are illustrated in Figure 1 (right).

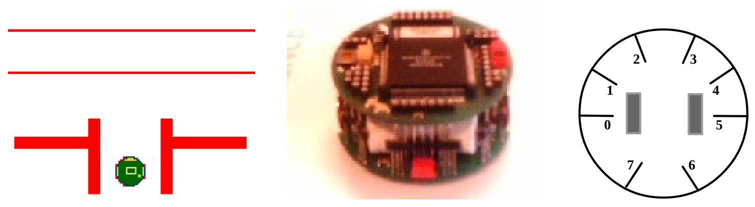


Figure 1: Left: experimental task. Right: the Khepera robot with eight infra-red distance sensors.

3 Results and analysis

We conducted 10 evolutionary runs consisting of 200 generations each. One of the 10 runs yielded an individual that solved the timing task. We examined the best

[2]Other strategies are also possible since genetic algorithms only optimize the fitness function rather than explicitely prescribing how to achieve a certain results. One other strategy might be that a robot runs in either direction and changes direction before stopping in the target region. Such a strategy also illustrates the capability of a robot to delay the action execution, but is unlikely to occur.

individuals of each evolutionary run and found robots which evolved a desired strategy (displaying their ability to delay their action execution), i.e., stopping after a learned amount of time steps.

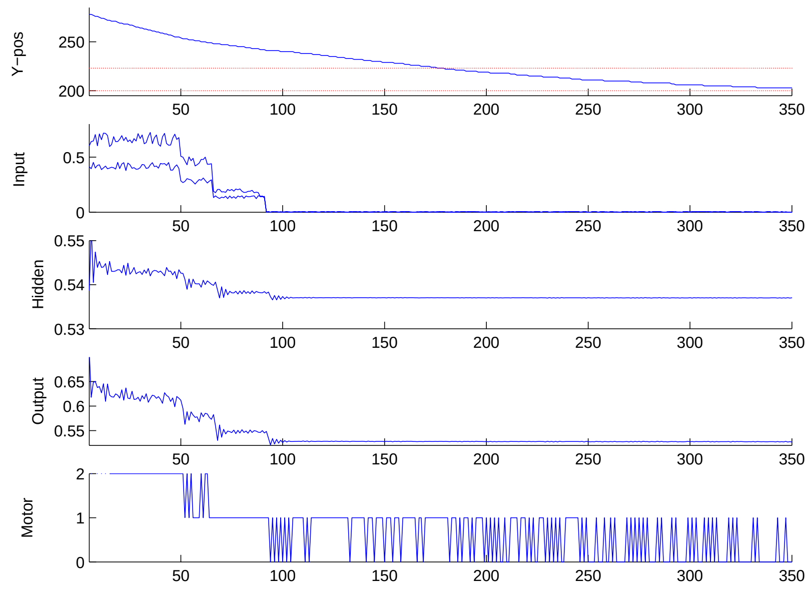
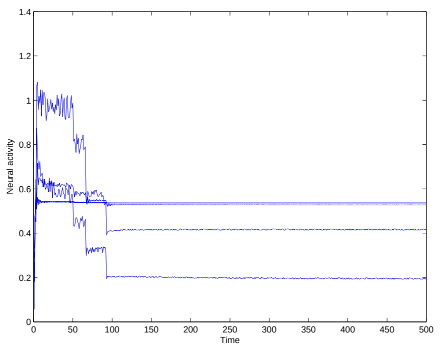


Figure 2: Illustration of the behaviour and internal dynamics of a robot solving the timing task. The five plots show (from top to bottom): (1) the vertical position of the robot as a function of time (the two horizontal parallel lines indicate the target region), (2) the sensory input as a function of time (for $t > 50$ the input is constant), (3) the activity of the hidden neuron and (4) output neuron as a function of time, and (5) the motor signal generated by the actuator as a function of time. (The test run lasted 500 time steps, but the last 150 steps no changes occur.)

One such robot following strategy (2), i.e. leaving the tunnel before decreasing its speed, is analysed. Figure 2 contains five plots illustrating the behaviour obtained by this robot. From top to bottom these plots illustrate (1) the robot's vertical position, (2) the normalized sensory input coming from the distance sensors, (3) the neural activity of the hidden neuron, (4) the neural activity of the output neuron, (5) the actual motor command issued to the robot. The top diagram illustrates that the speed of the robot decreases more and more when the robot approaches the end of the target region. It finally stops just in front of the (for the robot inperceptual) boundary of the target region. The speed decreases in three phases, related to the change of the input. The final decrease to zero speed takes place when the robot already entered the target region.

Figure 3 illustrates the neural activities and the synaptic weights over time. It can be observed that some synaptic weights change over a large amount of time



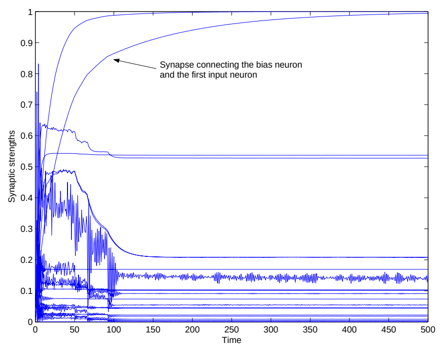
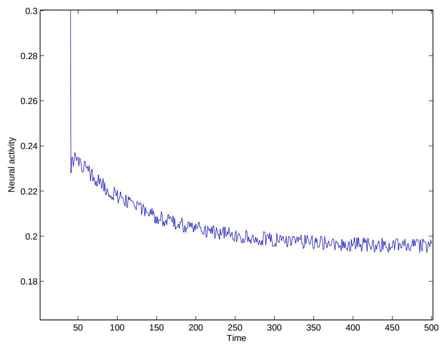


Figure 3: Left and right diagram illustrate the progression over time of the neurons (left) and synapses (right), respectively.



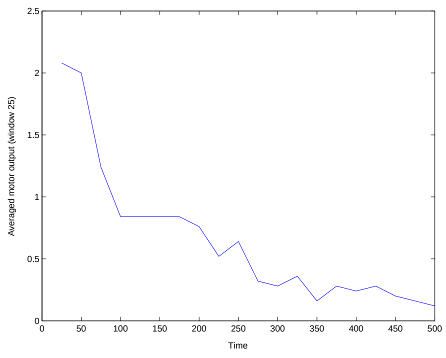


Figure 4: Left: detail of the progression of neural activity in the first input neuron. Over time, the bias neuron gains influence on the first input neuron. Right: averaged output speed. The average is taken over a time window of 25 steps. The decrease lasts as long as the decrease in activity of the first sensor neuron.

steps, i.e. they converge slowly to their equillibrium. The synapse requiring the longest amount of time to converge connects the bias neuron with the first input neuron. As a result, the neural activity of the input neuron decreases during the period of synaptic strength decay. The final outcome of the output neuron is strongly influenced by the first sensory neuron due to an inhibitory connection using the covariance learning rule with a high learning rate (0.9). The properties of the synapse connecting the input neuron and the output neuron ensure accurate response of the motor output to changes in the neural activity in the first input neuron (Figure 4, left). The slow changing dynamics of the neural substrate result in a continual decrease in speed (Figure 4, right). The robot has learned to delay its action execution.

Reference tests were conducted with the same controller to ensure strategy (2) was employed: regardless the initial position of the robot inside the tunnel, the robot has to stop after a learned amount of time. In this experiment, the robot started either at the end or beginning of the tunnel. It was observed that in both cases the robot needed 300 time steps to decrease its speed to null from the moment it left the tunnel (data not shown). This evolved strategy shows that the controller is either capable of integrating over persistent absence of stimuli [11] or has learnt how to *count* time steps. The result is equivalent: the robot delays its stopping action until a learned amount of time has passed.

4 Discussion and conclusion

Research in embodied cognitive science is shifting its emphasis from reactive agents to pro-active agents [8]. As a consequence, the focus is shifting from feed forward neural networks to more complex new neural networks that have some form of memory, such as the continuous Time Recurrent Neural Network (CTRNN) [1], the Dynamic Neural Network (DNN) [7], Time Delay Recurrent Neural Network (TDRNN) [3], Plastic Neural Network (PNN) [4]. These neural networks differ in their ability to deal with complex tasks. For example, several studies indicate that it is important for (neural) controllers to be able to exploit time dynamics [7, 10, 3, 11]. Open questions are whether we can explicate types of time dynamics that have to be exploited by successful pro-active agents, and whether we can design neural controllers that are able to exploit these dynamics. One type of time dynamics is the possibility for delaying the execution of actions. In our study, we showed that a PNN-controller can delay actions, so that it performs well on a timing task. The analysis suggests that the underlying mechanism facilitating the delay of actions is the slow change of synapse weights. Earlier findings [3, 11] also suggest that slow activation dynamics in a neural controller are required for performing timing tasks. Therefore, it seems that slow activation dynamics form one of the time dynamics necessary for successful proactive behaviour.

We started this study with the research question if PNNs can solve the timing task. The results show that a PNN is able to delay the action execution for solving the timing task. Analysis of the behaviour of the successfully evolved robot showed that slowly changing neural dynamics underlies this ability. Together with the results of other studies discussed in the previous section, we can conclude that neural networks can time action when they allow slow activation dynamics.

Acknowledgments

The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. Numerous colleagues are thanked for performing the real-time experiments on their personal computers.

References

[1] R.D. Beer. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):471–511, 1995.

[2] J. Blynel and D. Floreano. Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *Proceedings of the seventh international conference on simulation of adaptive behavior on From animals to animats*, pages 272 – 281, 2002.

[3] de Croon G., Nolfi S., and Postma E.O. *Toward pro-active embodied agents: On the importance of neural mechanisms suitable to process information in time.*

[4] D. Floreano and J. Urzelai. Evolutionary robotics with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–443, 2000.

[5] D. Floreano and J. Urzelai. Neural morphogenesis, synaptic plasticity, and evolution. *Theory in Biosciences*, 120:225–240, 2001.

[6] A.L. Nelson, E. Grant, J.M. Galeotti, and S. Rhody. Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and autonomous systems*, 46:179–173, 2004.

[7] S. Nolfi. Evolving robots able to self-localize in the environment: The importance of viewing cognition as the result of processes occurring at different time scales. *Connection Science*, 14:3:231–244, 2002.

[8] S. Nolfi. Power and limits of reactive agents. *Neurocomputing*, 49:119–145, 2002.

[9] S. Perretta and J.C. Gallagher. A general purpose java mobile robot simulator for artificial intelligence research and education. In *Proceedings of the 13th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS-2002)*, 2002.

[10] E. Tuci and M. Quinn. Behavioural plasticity in autonomous agents: a comparison between two types of controller. In *Proceedings of The Second European Workshop on Evolutionary Robotics EvoROB2003*, 2003.

[11] E. Tuci, V. Trianni, and M. Dorigo. “feeling” the flow of time through sensory-motor coordination. *Connection science*, 16(4):301–324, 2004.

[12] J. Urzelai and D. Floreano. Evolution of adaptive synpases: Robots with fast adaptive behavior in new environments. *Evolutionary Computation*, 9(4):495–524, 2001.