One-Shot Learning in the Road Sign Problem

Rafael C. Pinto*, Paulo M. Engel* and Milton R. Heinen†
*Informatics Institute
Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, CEP 91501-970, RS, Brazil
Email: {rcpinto,engel}@inf.ufrgs.br
†Center of Technological Sciences
Santa Catarina State University (UDESC), Joinville, CEP 89219-710, SC, Brazil
Email: miltonh@joinville.udesc.br

***Abstract*—In this work, a one-shot learning solution to the t-maze road sign problem is presented. This problem consists in taking the correct turning decision at a bifurcation after seeing a light signal some time steps before. The recently proposed Echo State Incremental Gaussian Mixture Network (ESIGMN) is used in order to learn the correct behavior after a single scan through a single training example (and its mirrored version) generated by a simple reactive controller. Experiments with different time delays between the signal and the decision point are performed, and the ESIGMN is shown to solve the problem while achieving good performance. This one-shot ability can be useful for online learning in robotics, since the robot can learn with minimum interaction with the environment.**

I. Introduction

The road sign problem consists in a task where an agent has to take some decision at some time step based on some signal observed in a previous time step [1]. A specific case is the t-maze with light signals, which, as the name suggests, consists of a t-shaped corridor with some light signal along the main segment. This light signal may appear on the left or right side of the corridor, indicating the correct side that the agent must turn when at the bifurcation (see figure 6). The main interest in this kind of task for robotics lays in the fact that the robot must retain the information from the light signal for some period of time until reaching the decision point (the bifurcation), thus requiring some type of memory. The longer the delay between the light signal and the decision point, the higher are the memory requirements. This is a simplification of a kind of problem that can vastly occur in the real world (it is analogous to traffic signs), and thus is of importance for the advancement of the field of robotics for complex environments and tasks.

In general, the preferred approach for tackling the road sign problem has been to use some kind of temporal neural network. In [1] an exponential trace memory of the network inputs was used as a temporal context added to the original inputs. In [2], it has been shown that recurrent neural networks can not handle very long delays between the signal and the decision point, but this has been shown later by [3] to be an issue with the training algorithm and not the recurrent architectures themselves, since it was possible to find solutions to the problem using recurrent neural networks trained by evolutionary algorithms. [4] solved the problem for arbitrary delays using higher level event extraction, and an Elman Network [5] was trained on this higher level data. More recently, Echo-State Networks (ESN) [6] also successfully solved the problem with reasonable accuracy [7] [8]. The main drawback of most of those approaches is that the used algorithms are iterative, meaning that the models are slowly learned according to some learning rate parameter, and many scans through the training set are necessary. In the case of the ESN, it is possible to train the model in an one-shot fashion by ordinary least squares, but it requires that all the training set must be collected beforehand.

In this work, we tackle the road sign problem using an Echo State Incremental Gaussian Mixture Network (ESIGMN) [9] [10], which combines the reservoir computing (RC) [11] approach of ESNs with the Incremental Gaussian Mixture Network (IGMN) [12] [13]. This allows us to learn the desired behavior from a single scan through a training example and its mirrored version and does not need a separate learning phase before being put into operation. This ability can be useful for online learning in robotics, since the robot can learn incrementally with minimum interaction with the environment.

The rest of this work is structured as follows: in section II, the IGMN algorithm is described in detail. Section III describes the ESIGMN algorithm as an extension for the IGMN. Section IV will describe the experiments in which the ESIGMN was tested and corresponding results. Section V finishes this work with concluding remarks.

II. Incremental Gaussian Mixture Network

The Incremental Gaussian Mixture Network (IGMN) [13] is a supervised algorithm that uses as its base an incremental approximation of the EM algorithm [14], the IGMM (Incremental Gaussian Mixture Model) [15]. It creates and continually adjusts a probabilistic model consistent to all sequentially presented data, after each data point presentation, and without the need to store any past data points. Its learning process is aggressive, or "one-shot", meaning that only a single scan through the data is necessary to obtain a consistent model.

IGMN adopts a Gaussian mixture model of distribution components (known as a *cortical region*) that can be expanded to accommodate new information from an input data point, or reduced if spurious components are identified along the learning process. Each data point assimilated by the model contributes to the sequential update of the model parameters

U.S. Government work not protected by U.S. copyright

based on the maximization of the likelihood of the data. The parameters are updated through the accumulation of relevant information extracted from each data point.

Differently from IGMM, however, the IGMN is capable of supervised learning, simply by assigning any of its input vector elements as outputs (any element can be used to predict any other element, like autoassociative neural networks [16]). This architecture is depicted on figure 1. Next subsections describe the algorithm in more detail (however, we found that some aspects of the IGMM are useful for the IGMN, so the algorithm will be slightly modified to recover those aspects).

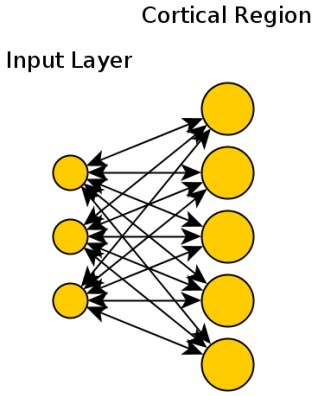


Fig. 1. An example of IGMN with 3 input nodes and 5 Gaussian components. Any input element can be predicted by using any other element, which means that the input vector can actually be divided into input and output elements.

A. Learning

The algorithm starts with no components, which are created as necessary (see subsection II-B). Given input $\mathbf{x}$ (a single instantaneous data point), the IGMN algorithm processing step is as follows. First, the likelihood for each component j is computed:

$$\bar{p}(\mathbf{x}|j) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right) \tag{1}$$

$$p(\mathbf{x}|j) = \frac{\bar{p}(\mathbf{x}|j)}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \tag{2}$$

where D is the input dimensionality, $\boldsymbol{\mu}_j$ the j^{th} component mean, $\mathbf{C}_j$ its covariance matrix and $\bar{p}(\mathbf{x}|j)$ is the the likelihood before normalization (each value of $\bar{p}(\mathbf{x}|j)$ will be in the interval [0 ,1], with 1 being a perfect match between $\mathbf{x}$ and $\boldsymbol{\mu}_j$). If any $\bar{p}(\mathbf{x}|j)$ is greater than a threshold τ_{max} (e.g., 0.999), it is assumed that the model already represents well the current data point $\mathbf{x}$, and therefore no learning takes place. Besides speeding up the algorithm in some cases, this condition avoids excessive shrinking of the Gaussian components when very similar data points are seen many times (e.g., sonar data along a long corridor), which could cause numerical problems as well as disturbing the efficiency of the novelty criterion used for component creation (a similar concept was introduced in [17] and is not part of the original IGMN algorithm).

Supposing that the learning procedure goes on, posterior probabilities are calculated for each component as follows:

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{q=1}^{M} p(\mathbf{x}|q)p(q)} \quad \forall j \tag{3}$$

where M is the number of components. Now, parameters of the algorithm must be updated according to the following equations:

$$v_j(t) = v_j(t-1) + 1 \tag{4}$$

$$sp_j(t) = sp_j(t-1) + p(j|\mathbf{x}) \tag{5}$$

$$\mathbf{e}_j = \mathbf{x} - \boldsymbol{\mu}_j \tag{6}$$

$$\omega_j = \frac{p(j|\mathbf{x})}{sp_j} \tag{7}$$

$$\Delta\boldsymbol{\mu}_j = \omega_j\mathbf{e}_j \tag{8}$$

$$\boldsymbol{\mu}_j(t) = \boldsymbol{\mu}_j(t-1) + \Delta\boldsymbol{\mu}_j \tag{9}$$

$$\mathbf{C}_j(t) = \mathbf{C}_j(t-1) - \Delta\boldsymbol{\mu}_j\Delta\boldsymbol{\mu}_j^T + \omega\left[\mathbf{e}\mathbf{e}^T - \mathbf{C}_j(t-1)\right] \tag{10}$$

$$p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q} \tag{11}$$

where sp_j and v_j are the accumulator and the age of component j, respectively, and $p(j)$ is its prior probability.

B. Creating New Components

If instead of having any $\bar{p}(\mathbf{x}|j)$ greater than a threshold τ_{max}, we have all $\bar{p}(\mathbf{x}|j)$ below some threshold τ_{min} (e.g., 0.001) (or there are no components) and a stability criterion is satisfied, which means having all v_j greater than some age_{min} (e.g., $D+1$, where D is the input space dimensionality), then a new component j is created and initialized as follows:

$$\boldsymbol{\mu}_j = \mathbf{x}; \quad sp_j = 1; \quad v_j = 1; \quad p(j) = \frac{1}{\sum_{i=1}^{M} sp_i}; \quad \mathbf{C}_j = \sigma_{ini}^2\mathbf{I}$$

where M already includes the new component and σ_{ini} can be obtained by:

$$\sigma_{ini} = 3\delta std(\mathbf{x}) \tag{12}$$

where δ is a manually chosen scaling factor (e.g., 0.01) and std is the standard deviation of the dataset (the constant 3 is

chosen to better cover the full range of values in the dataset according to a Gaussian distribution). Note that the IGMN is an online and incremental algorithm and therefore it may be the case that we do not have the entire dataset to extract descriptive statistics. In this case the standard deviation can be just an estimation, without impacting the algorithm.

Using this threshold based on the likelihoods of each component is the method used in the IGMM which was discarded in favor of an error-based threshold. We found the original IGMM threshold to be more robust and adaptive (i.e., it works better within distributions that vary too much) than the error threshold, and therefore it is reintroduced into the IGMN here. The stability criterion was also removed in the IGMN when coming from the IGMM, but we empirically found it to be a valuable tool for removing sensitivity from the δ parameter (the initial size of the newly created Gaussian components), which now can be initialized with a small value without resulting in an excessive creation of components. Using the standard deviation instead of the range / amplitude of the data set (as in the original algorithm) also has benefits when the different input dimensions have different distributions, allowing for a more robust learning procedure.

C. Removing Spurious Components

A component j is removed whenever $v_j > v_{min}$ and $sp_j < sp_{min}$, where v_{min} and sp_{min} are manually chosen (e.g., 5.0 and 3.0, respectively). In that case, also, $p(q)$ must be adjusted for all $q \in M$, $q \neq j$, using (11). In other words, each component is given some time v_{min} to show its importance to the model in the form of an accumulation of its posterior probabilities sp_j.

D. Recalling

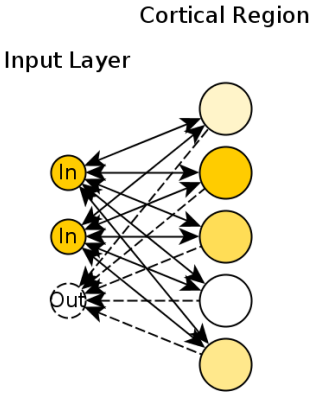


Fig. 2. An example of IGMN with 3 input nodes and 5 Gaussian components. Two of the input elements were selected for estimating the third one. The different color intensities inside each Gaussian component represent their different posterior probabilities after seeing data $\mathbf{x}_i$ (only the given elements), and are used to weight the contributions of each component to the final result.

In IGMN, any element can be predicted by any other element, and this is the biggest addition in relation to the unsupervised IGMM algorithm. This is done by reconstructing data from the target elements ($\mathbf{x}_t$, a slice of the entire input vector $\mathbf{x}$) by estimating the posterior probabilities using only the given elements ($\mathbf{x}_i$, also a slice of the entire input vector $\mathbf{x}$), as follows:

$$p(j|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|j)p(j)}{\sum_{q=1}^{M} p(\mathbf{x}_i|q)p(q)} \quad \forall j \tag{13}$$

It is similar to (3), except that it uses a modified input vector $\mathbf{x}_i$ with the target elements $\mathbf{x}_t$ removed from calculations. After that, $\mathbf{x}_t$ can be reconstructed using the conditional mean equation:

$$\hat{\mathbf{x}_t} = \sum_{j=1}^{M} p(j|\mathbf{x}_i)(\boldsymbol{\mu}_{j,t} + \mathbf{C}_{j,ti}\mathbf{C}_{j,i}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{j,i})) \tag{14}$$

where $\mathbf{C}_{j,ti}$ is the submatrix of the jth component covariance matrix associating the unknown and known parts of the data, $\mathbf{C}_{j,i}$ is the submatrix corresponding to the known part only and $\boldsymbol{\mu}_{j,i}$ is the jth's component mean without the element corresponding to the target element. The recalling procedure is depicted in figure 2.

III. Echo-State Incremental Gaussian Mixture Network (ESIGMN)

Reservoir Computing (RC) [11] is a recently coined term for a not so recent neural pattern processing paradigm where a random, non-linear, fixed and large hidden layer with recurrent connections, called a reservoir, is used as an excitable medium where interesting dynamic features of the data stream can be extracted. It is similar to a random filter bank, producing transformations over input data. Albeit having fixed reservoir weights, the reservoir output states are sufficient to successfully train linear regression / classification algorithms on non-linear dynamic tasks, thus potentially turning any static linear algorithm into a non-linear dynamic one (the larger the number of neurons in the reservoir, potentially longer is the memory). The prominent example of RC in computer science is the Echo State Network (ESN) [6], which uses a sigmoidal reservoir layer to give nonlinear dynamic capabilities to a linear regression output layer. A typical ESN architecture can be seen in figure 3.

In order to have a stable reservoir, however, the echo state property must be assured. It means that when a null vector is continually fed into the reservoir as its input, its state vector must decay to a null vector too, as time tends towards infinity. In practical terms, this can be ensured by rescaling the largest absolute eigenvalue $|\lambda_{max}|$ of the reservoir recurrent weight matrix $\mathbf{W}$, i.e. its spectral radius, to a value in the interval (0, 1).

Using the concept of RC, the Echo State Incremental Gaussian Mixture Network (ESIGMN) [9] augments the IGMN with an ESN-style reservoir between its input and cortical region. The reservoir is responsible for mapping the input space into a feature space which captures temporal dynamics

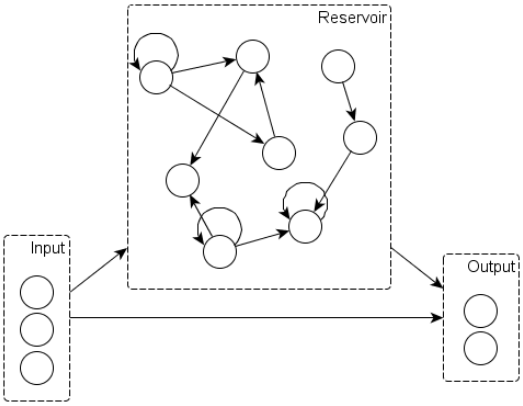


Fig. 3. An example of ESN with 3 inputs, 2 targets/outputs and 8 reservoir neurons.

of the data, as with the ESN. But instead of feeding the input and the reservoir state into a linear output layer, they are fed into an unmodified IGMN in the ESIGMN algorithm. The IGMN does its usual spatial processing, but its inputs already incorporate temporal information. Feedback connections from the outputs to the reservoir are also possible and will be used in this work. This architecture can be seen in figure 4.

Therefore, all IGMN equations from section II apply, with a difference only in the source of the IGMN input. In ESIGMN, the input data $\mathbf{x}$ is divided into input $\mathbf{x}_i$ and output/target elements $\mathbf{x}_t$ as with the IGMN, and is processed in the following way:

$$\mathbf{s}(t) = f(\mathbf{W}_{in,r}\mathbf{x}_i(t) + \mathbf{W}\mathbf{s}(t-1)) \tag{15}$$

$$\bar{\mathbf{x}}(t) = [\mathbf{s}(t); \mathbf{x}(t)] \tag{16}$$

where $\mathbf{s}$ is the reservoir state, $\mathbf{x}_i$ is the known part of the input (the actual input portion of the data), excluding the target values $\mathbf{x}_t$, $\mathbf{W}_{in,r}$ is the input-to-reservoir weight matrix (excluding the target elements), and f is an activation function, usually the logistic or the hyperbolic tangent functions. The IGMN portion of the architecture is trained by receiving $\bar{\mathbf{x}}$ (the concatenation of the full input and reservoir state) as its input. By omitting $\mathbf{x}_t$ and using (13) for recalling, it is possible to predict the output $\mathbf{x}_t$.

IV. Experiments and Results

The ESIGMN was used to learn trajectories for the road sign problem in a t-maze with different signal delays. The KiKS [18] simulator was used for this task. It is a Matlab Khepera simulator which supports the 8 infrared proximity sensors of the real robot (although only the 6 front sensors are used here), 8 luminosity sensors (only the 2 lateral ones are used here) and allows for the placement of light sources on maps. It also automatically adds noise to all sensors and actuators based on a model of the real robot. A snapshot of the simulator can be seen in figure 5, also showing the sensor and actuator layout of the Khepera robot.

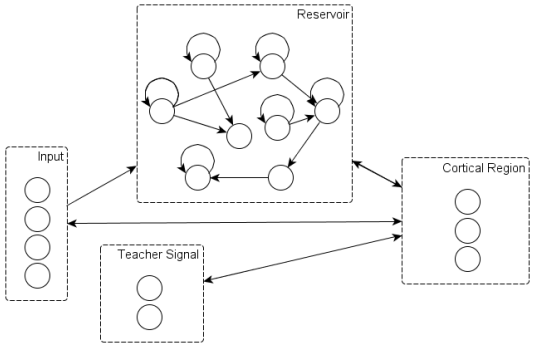


Fig. 4. An example of ESIGMN with 4 inputs ($\mathbf{x}_i$), 2 targets/outputs ($\mathbf{x}_t$), 3 Gaussian components and 8 reservoir neurons. Only the actual input portion of the data is used to update the reservoir, while the full input data (with target values / teacher signal) is fed into the IGMN together with the reservoir state $\mathbf{s}$ in order to train it. By omitting $\mathbf{x}_t$, the IGMN can be used in recalling mode to predict $\mathbf{x}_t$.

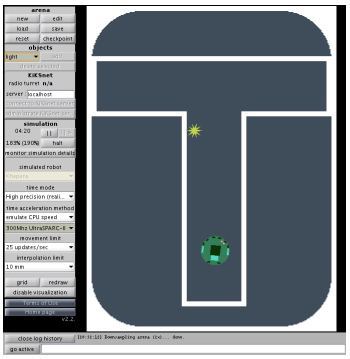
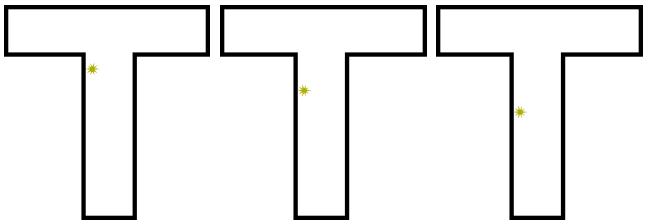


Fig. 5. The KiKS Khepera simulator running the t-maze task with a light signal to the left (nearest position).



(a) Left-Near (b) Left-Middle (c) Left-Far

Fig. 6. T-maze with different positions for the light signal on the left side. Signals on the right side are just mirrored versions of the left ones.

	Delay
Near	10.0 (2.0)
Middle	29.5 (5.5)
Far	51.5 (5.0)

TABLE I
Distance from light signal to decision point in time steps for each t-maze configuration. Medians outside parenthesis, MADs inside.

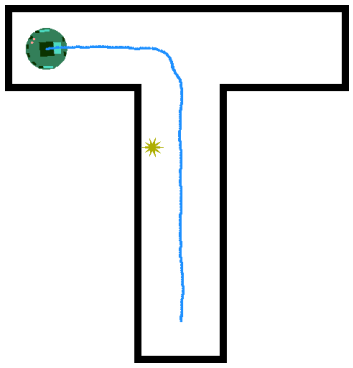


Fig. 7. The single training data used for one of the light signal positions.

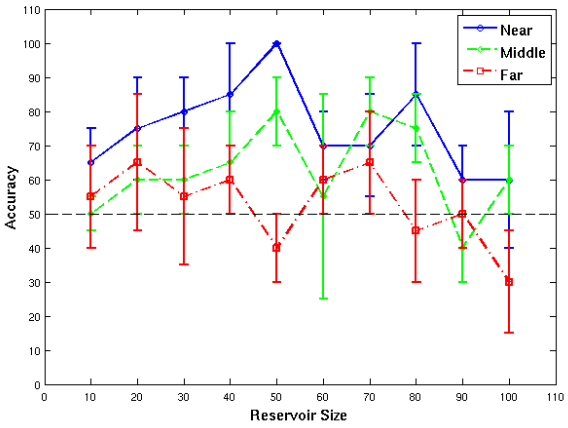


Fig. 8. Results of the experiments with the 3 different positions of the light signal, with reservoir sizes from 10 to 100. The black dashed horizontal line represents the accuracy given by random guessing (50%).

Reservoir Size	Near	Middle	Far
10	65 (10)	50 (5)	55 (15)
20	75 (15)	60 (10)	65 (20)
30	80 (10)	60 (10)	55 (20)
40	85 (15)	65 (15)	60 (10)
50	100 (0)	80 (10)	40 (10)
60	70 (10)	55 (30)	60 (10)
70	70 (15)	80 (10)	65 (15)
80	85 (15)	75 (10)	45 (15)
90	60 (10)	40 (10)	50 (10)
100	60 (20)	60 (10)	30 (15)

TABLE II
ACCURACY (%) OBTAINED FOR EACH POSITION OF THE LIGHT SIGNAL WITH DIFFERENT RESERVOIR SIZES. MEDIANS OUTSIDE PARENTHESIS, MADS INSIDE.

A. Setup

A single trajectory example and its mirrored version were used to train the algorithm for a single epoch (an example can be seen in figure 7). The example trajectories were programmed by hand as a reactive controller (with privileged information about the correct turning directions) and Gaussian noise was added to sensors and actuators, scaled by 10% of the range of each one (in addition to the noise model already used by the simulator). Twelve-dimensional vectors ($\mathbf{x}$) were used as inputs for the ESIGMN, consisting of 2 current motor speeds, 2 current light sensor intensities, 6 front proximity values (summing up to a 10-dimensional input $\mathbf{x}_i$ vector) and 2 motor speed outputs (target/output $\mathbf{x}_t$). Gaussian noise was added to all these values, scaled to 5% of each sensor/actuator range. Reservoir sizes varied from 10 to 100, while the spectral radius of the reservoir was fixed at 0.99 for all experiments. Input-to-reservoir weight scaling was set to 0.1. Some random leaking factor in the (0,1) interval was added to each reservoir neuron in order to improve memory. The other ESIGMN parameter values were the ones suggested in section II. Some experiments were done in order to find good values for all aforementioned hyperparameters, but no optimality is claimed. No regularization was done, i.e., no noise was added to reservoir outputs, which means it is possible to overfit the data. As can be seen, there is a lot of room for improvement, which will be verified in future works, but here the goal was only to show the one-shot learning capability of the proposed approach. As for the environment, light signal delays varied from approximately 10 to 50 time steps, each with its own training set, and the robot was tested only in the task corresponding to the training set used. Those different positions of the light signal can be seen in figure 6 and their estimated delays can be seen in table I in the form of medians and MADs (Median Absolute Deviations from the Median), according to data sampled from 10 runs of the reactive controller (it is important to note that, unlikely with some previous works on the subject, training data was not downsampled to give the reservoir an artificial longer memory). The full maze size in time steps, estimated in the same way, has a median of 257.5 with a MAD of 4.5. Results will also be described in terms of medians and MADs, since they are more robust descriptive statics than the mean and standard deviation.

B. Results

As can be seen in figure 8 and table II, the ESIGMN could find a solution to the t-maze road sign problem with just 1 training example trajectory (for each side) and 1 epoch, given sufficient neurons in the reservoir. Moreover, its learned trajectories are smoother (figure 9) and faster than the original one (median of 148 time steps for the full maze). For both the ”near” and ”middle” light signal positions, the optimal reservoir size was 50, which gave 100% and 80% accuracy, respectively. The best accuracy obtained for the ”far” position was 65%, which albeit being a poor result, is still better than random guessing, which suggests that some of the generated reservoirs could keep at least small memory traces of the light signal. Those results are not symmetric, with a slight preference for the left side, which might be due to a small displacement of the robot from the exact center of the corridor at its starting position.

From those results, it can be seen that the accuracy falls after increasing the reservoir by a certain amount. This can be attributed to 2 factors: overfitting and slow response time. The

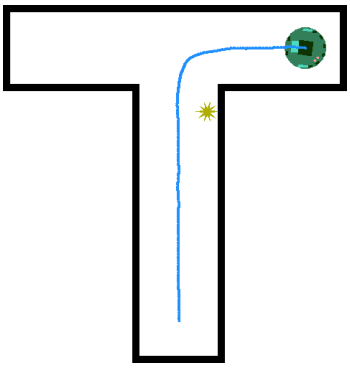


Fig. 9. A trajectory learned by the ESIGMN algorithm.

later is due to the $O(n^3)$ time complexity of matrix inversions inside each Gaussian component (where n is the input size), which makes ESIGMNs with large reservoirs very slow (since the reservoir states are concatenated to input signals).

Also, to exclude the hypothesis that some slight light stimulus could be guiding the robot reactively (instead of using memory), the static IGMN was tested on the task too. It could do no better than random guessing, obtaining 50% accuracy for all signal positions.

It is interesting to note that if tapped-delay lines were used instead of a reservoir [10], at least the length of the signal-decision interval should be used as the length of the tapped-delay lines, approximately 10, 30 and 50 for the near, middle and far positions respectively. For the 10-dimensional original inputs used, it would mean 110, 310 and 510 total inputs to the IGMN for the three cases, way beyond the 60, 60 and 80 (input + reservoir) required by the reservoir approach. Even if only the two light sensors are included in the tapped-delay lines, the number of inputs of the three cases would be 30 (10 original inputs plus 20 new ones), 70 and 110, respectively, which is advantageous only for small signal-decision lengths. On the other hand, the memory of the light sign would always be perfectly remembered at the decision point with tapped-delay lines, supposing that at least the minimum delay is used for each position. Such hypothesis is yet to be verified experimentally and both approaches to be compared and are left for future works in order to focus on the reservoir approach in this work.

V. Conclusions

The ESIGMN algorithm could find solutions the t-maze road sign problem efficiently after a single scan through a single training example and its mirrored version. By using a reservoir with 50 neurons, it was able to perfectly learn the task with an approximate delay of 10 time steps between the signal and the decision point. It can also handle larger delays with some degraded performance. A drawback of the proposed solution is the time complexity involved in matrix inversions, which inviabilize the use of the algorithm for larger reservoir sizes, which in consequence limits its potential for longer memories. It is yet to be verified how long a memory can be in this algorithm by increasing the reservoir, but it will require some modification of the algorithm to avoid matrix inversions or test it in more powerful computers.

The results presented in this work are relevant to the field of robotics, since they can enable a robot to learn memory tasks in partially observable environments with minimal physical interaction, reducing training time and potential damage. Here, the training was done by means of supervised learning of an example trajectory, but more autonomous solutions will be investigated, including goal seeking and reinforcement learning.

References

[1] C. Ulbricht, "Handling time-warped sequences with neural networks," in *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996, pp. 180–189.

[2] R. Rylatt and C. Czarnecki, "Embedding connectionist autonomous agents in time: The 'road sign problem'," *Neural Processing Letters*, vol. 12, no. 2, pp. 145–158, 2000.

[3] M. Thieme and T. Ziemke, "The road sign problem revisited: handling delayed response tasks with neural robot controllers," *Hallam et al.[2]*, pp. 228–229, 2002.

[4] F. Linaker and H. Jacobsson, "Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond," in *Internetional Joint Conference on Artificial Intelligence*, vol. 17, no. 1, 2001, pp. 777–782.

[5] J. Elman, "Finding structure in time," *Connectionist Psychology: A Text with Readings*, 1999.

[6] H. Jaeger, "The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'," Technical Report GMD Report 148, German National Research Center for Information Technology, Tech. Rep., 2001.

[7] E. Antonelo, B. Schrauwen, and D. Stroobandt, "Experiments with reservoir computing on the road sign problem," in *Proceedings of the VIII Brazilian Congress on Neural Networks (CBRN), Florianopolis. 2007*, 2007.

[8] ——, "Mobile robot control in the road sign problem using reservoir computing networks," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 911–916.

[9] R. Pinto, P. Engel, and M. Heinen, "Echo State Incremental Gaussian Mixture Network for Spatio-Temporal Pattern Processing," in *Proceedings of the IX ENIA - Brazilian Meeting on Artificial Intelligence*, Natal (RN), jul 2011.

[10] R. C. Pinto, "Online incremental one-shot learning of temporal sequences," Master's thesis, Universidade Federal do Rio Grande do Sul (UFRGS), dec 2011.

[11] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[12] M. Heinen, P. Engel, and R. Pinto, "IGMN: An Incremental Gaussian Mixture Network that Learns Instantaneously from Data Flows," in *Proceedings of the IX ENIA - Brazilian Meeting on Artificial Intelligence*, Natal (RN), jul 2011.

[13] M. Heinen, "A connectionist approach for incremental function approximation and on-line tasks," Ph.D. dissertation, Universidade Federal do Rio Grande do Sul - Instituto de Informática, 2011.

[14] A. Dempster, N. Laird, D. Rubin *et al.*, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[15] P. Engel and M. Heinen, "Incremental learning of multivariate gaussian mixture models," *Advances in Artificial Intelligence–SBIA 2010*, pp. 82–91, 2011.

[16] D. Rumelhart and J. McClelland, *Parallel distributed processing*. MIT Pr., 1986.

[17] C. Northfleet, "Aprendizado on-line e formação incremental de conceitos em um controlador híbrido para futebol de robôs," Undergraduate Thesis, dec 2011.

[18] T. Nilsson, "Kiks is a khepera simulator," *Stockholm, Sweden, Umea University*, 2001.