Local Structure Helps Learning Optimized Automata in Recurrent Neural Networks

Jonathan Binas, Giacomo Indiveri, and Michael Pfeiffer
Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland
Email: jbinas@ini.ethz.ch

***Abstract*—Deterministic behavior can be modeled conveniently in the framework of finite automata. We present a recurrent neural network model based on biologically plausible circuit motifs that can learn deterministic transition models from given input sequences. Furthermore, we introduce simple structural constraints on the connectivity that are inspired by biology. Simulation results show that this leads to great improvements in terms of training time, and efficient use of resources in the converged system. Previous work has shown how specific instances of finite-state machines (FSMs) can be synthesized in recurrent neural networks by interconnecting multiple soft winner-take-all (SWTA) circuits — small circuits that can faithfully reproduce many computational properties of cortical networks. We extend this framework with a reinforcement learning mechanism to learn correct state transitions as input and reward signals are provided. Not only does the network learn a model for the observed sequences, and encode it in the recurrent synaptic weights, it also finds solutions that are close-to-optimal in the number of states required to model the target system, leading to efficient scaling behavior as the size of the target problems increases.**

I. Introduction

Finite automata, or finite-state machines (FSMs), present one of the simplest frameworks to model deterministic sequential behavior, and are essential building blocks in theoretical computer science [1]. FSMs can model many aspects of high level deterministic behavior, such as production of movement sequences, navigation, state-dependent decision making, logical reasoning, or understanding and production of language. Although many neural processes can be better modeled by probabilistic graphical models, taking into account the inherent environmental and neural stochasticity [2], [3], almost deterministic sequences of neural activation have been observed in brains of various species and during diverse activities. Examples include synfire chains [4], sequences during song production in birds [5], or location-dependent patterns during navigation in rats [6].

It has been shown previously how FSM-like dynamics can be realized in networks of interconnected neural populations [7], [8], [9]. In these implementations, states are represented as point attractors in a multistable attractor neural network. Special gating units implement a mechanism for switching between different states conditional on the current state and the external input stimulus. This framework allows bottom-up engineering of given finite automata by setting the relevant connections in the network, i.e. the transition table of a given automaton can directly be translated to connections between populations. The described model is particularly appealing as it is based solely on interconnected winner-take-all networks, small circuits that closely match connectivity patterns found in neuroanatomy [11], [12].

While those previous approaches have addressed the synthesis of state machines with known dynamics, this article presents a biologically plausible neural framework for learning deterministically behaving systems in the form of neural FSMs. We extend the work initiated in [7] by developing a reinforcement learning mechanism, such that an agent equipped with our model can learn and adapt its deterministic behavior through sparse, external reward signals. Specifically, the network produces an output after receiving a sequence of inputs and is provided feedback in terms of a reward signal that indicates whether the output corresponds to the target behavior. The network then adapts its internal model using a reward-modulated Hebbian-type learning rule [13], [14] to maximize the correspondence of its behavior with the target behavior.

For any neural system, the number of states required to model the target behavior cannot be known in advance. It is thus crucial that the system makes efficient use of the available neural resources, since this can speed up learning and improve the generalization capabilities. We find that within our framework simple, local modulation of the network structure leads to greatly improved solutions in terms of the optimality of the solutions that are learned, as well as in the search time required to find a valid solution. Here, the term optimal refers to the minimal automaton that implements the desired dynamics. Our structural assumptions nicely translate to real biological systems, where nearby populations of neurons are more likely to be connected than far away ones.

In this article, we introduce our system first as a formal model and then map it to a network of neural populations. We illustrate how local structure in the network connectivity leads to solutions that are globally optimized in terms of the number of state populations used to implement a target behavior and dramatically improves the learning performance.

II. Definition of the Model

An FSM is defined by a set of states $\boldsymbol{S} = \{1, 2, 3, \ldots\}$, an alphabet of input symbols $\boldsymbol{A} = \{a, b, c, \ldots\}$, and a transition table $\boldsymbol{T}$ that contains an entry $T_i^k = j$ for every state i and input symbol k, specifying a transition to a target state j for that particular state-input combination. Furthermore, there is an initial state s_0, and a set of one or multiple *accept* states $\mathbf{S}_{\text{acc}} \subseteq \mathbf{S}$. The system carries out some computation by switching between its internal states while processing a finite string of input symbols. The outcome is binary – the system either ends up in one of the accept states in $\mathbf{S}_{\text{acc}}$ or not. This mechanism can be used to parse and classify given input strings, i.e. to make a decision on the basis of a sequence of input stimuli. An illustration of a simple FSM is shown in fig. 1A. The system is initialized in state 1 and ends up in the accept state 3 if a number of blue inputs is provided, followed by a number of red inputs. Note that self-transitions are not shown, but are assumed implicitly for all inputs for which no transition is specified.

A. *Neural Populations as Computing Elements*

We illustrate the implementation of finite automata in attractor neural networks by means of an abstract population model, which we use to introduce the learning rule and analyze various aspects of the plastic system.

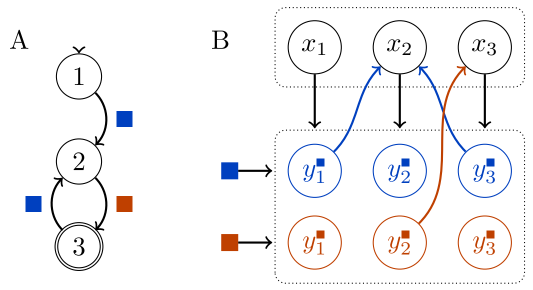

Fig. 1. Building state machines from neural populations. A) Abstract directed graph representation of an example state machine, consisting of three states (1, 2, 3) and two input symbols (red, blue). Arrows indicate state transitions, double circles indicate accepting states. B) Implementation of the state machine from A) in a neural network of coupled WTA circuits (dashed boxes). The units X_i in the top WTA represent the states, while the units y_j^k of the lower WTA trigger transitions between them. Each unit in the lower WTA corresponds to one particular input-state combination, indicated by color and state index, and can activate its target state through a strong excitatory projection.

Our implementation is based on two interconnected WTA networks (fig. 1B), one of which serves as memory of the current state while the other one functions as a gating mechanism, activating exactly one transition per state-input combination. This formalism was introduced by [7]. A WTA network contains a number of neural populations that compete through mutual inhibition, such that only one of them can be active at a time. If the parameters are identical for all populations, the active population of a WTA is the one receiving the greatest input. If local self-excitation is added to the circuit, i.e. each population recurrently excites itself, the populations can retain their activity, and therefore a memory, even if no external input is provided. However, due to the mutual inhibition only one population is active at a time, such that one can switch from one attractor state to another by providing a strong input to the other population. A WTA network with recurrent excitation is used to represent the internal states, that is, activation of population x_i indicates that the system is in state i and this state is retained even if no input is provided.

A second WTA network without recurrent excitation implements the transitions between states. This second WTA contains exactly one population for every state-input symbol combination, whereby each population receives input from one input symbol and one of the state populations. Due to the competition mechanism, only the population y_j^k corresponding to the current state j and the current input symbol k will become active. An activation threshold makes sure that the populations become active only if they receive both input from a state population and an input symbol and are silent otherwise. In order to implement a transition to a target state, the population corresponding to a particular state-input symbol pair simply needs to provide a strong input to the population representing the target state. Furthermore, a random, non-empty subset of the state populations $\mathbf{X}_{\text{acc}} \subseteq \mathbf{X}$ is defined as the set of accepting states, while all other state populations correspond to reject states. One state population x_0 is defined to represent the initial state.

The corresponding population-based implementation of the FSM illustrated in fig. 1A is shown in fig. 1B. It is discussed elsewhere [7], [9] how the neuron parameters have to be chosen to implement such dynamics in networks of threshold-linear units. In this study, for simplicity, we assume the activation of a unit to be either 0 or 1 and do not consider transient dynamics.

B. *The Learning Rule*

Rather than manually specifying the transitions, the idea behind this work is to learn them based on sequential sensory input and reinforcement signals. A neural network of the type described above is initialized with transition weights set to random values in $(w_{\min}, w_{\max})$ where $w_{\min}$ and $w_{\max}$ are chosen such that the input to state populations is large enough to switch to a different state. The training is based on a number of trials, each of which consists of an input sequence of length N_i and a reward signal r_i. At the beginning of a trial i, the agent is set to its initial state, i.e. the population x_0 that was specified to correspond to the initial state is activated. The trial length N_i is then chosen from a uniform distribution over $[1, N^{\max}]$ and N_i randomly chosen input symbols are provided at discrete times $t_1, \ldots, t_{N_i}$ for a period τ_0 each. At the end of the sequence, it is checked whether the agent is in an accept state or not and whether this outcome corresponds to the outcome of the target automaton if provided the same sequence of inputs. If the outcomes are in agreement, a reward $r_i = 1$ is provided to the agent. If not, a negative reward signal (or punishment) $r_i = -1$ is provided.

As we intend to learn the right transitions, the learning rule outlined in the following solely needs to alter the 'transition connections', i.e. the connections from the gating populations to the state populations. All other connections remain fixed, although they could in principle also be learned with biologically plausible rules [15]. Due to the competition mechanism, only one of the state populations will be active after an input symbol is provided and a gating population has been activated, namely the one receiving the greatest input from that gating population.

We denote by w_{ij}^k the connection from the gating population y_j^k to the state population x_i. While inputs are provided and the system switches between internal states, each transition connection w_{ij}^k locally records the traces Γ_{ij}^k and $\tilde{\Gamma}_{ij}^k$, which are updated after every input presentation as follows:

$$\Gamma_{ij}^k \mapsto (1-\lambda)\Gamma_{ij}^k + \lambda y_j^k(t_n) x_i(t_n + \tau_0)\,, \qquad (1)$$

$$\tilde{\Gamma}_{ij}^k \mapsto (1-\lambda)\tilde{\Gamma}_{ij}^k + \lambda y_j^k(t_n)\left(1 - x_i(t_n + \tau_0)\right)\,, \qquad (2)$$

where λ is a constant between 0 and 1 (in all simulations we set $\lambda = 1/3$) that determines the speed of decay of the traces. Without loss of generality, we can assume that the activation of a population is either 1 (active) or 0 (silent). Thus, the quantity Γ_{ij}^k is increased whenever the pre- and postsynaptic populations of the corresponding connection w_{ij}^k are co-active, while $\tilde{\Gamma}_{ij}^k$ is increased when only the presynaptic population y_j^k is active.

At the end of the trial, the reward signal $r \in \{-1, 1\}$ is provided and the weights are updated proportional to the learning rate η (set to 1 in all simulations) as follows:

$$w_{ij}^k \mapsto w_{ij}^k + r\eta \left(g_r(w_{ij}^k)\Gamma_{ij}^k - \tilde{g}_r(w_{ij}^k)\tilde{\Gamma}_{ij}^k \right), \tag{3}$$

where $\tilde{g}_r = g_{-r}$ and

$$g_r(w_{ij}^k) = \begin{cases} w_{\max} - w_{ij}^k & \text{if } r = 1, \\ w_{ij}^k - w_{\min} & \text{if } r = -1. \end{cases} \tag{4}$$

The weight dependence g_r ensures that the minimum and maximum weight values are never exceeded. Overall, the update can be regarded as a reward-modulated *Hebbian* learning rule. This is evident when noting that either Γ_{ij}^k or $\tilde{\Gamma}_{ij}^k$ (or both) are zero for a given connection after the trial. This is the case because weights are not updated during the trial, so a transition is either taken always or never. If the reward is positive, connections whose pre- and postsynaptic populations have been active together are strengthened. On the other hand, connections which do not lead to activation of a postsynaptic population are weakened. If the reward is negative, the update is inverted, i.e. the connections corresponding to transitions that took place are weakened, while others are strengthened. Thus, over many trials, the connections should converge to values that lead to the maximum reward and therefore always fulfill the target dynamics.

C. Local Structure in the Network Topology

In a realistic scenario, the number of available state populations will not exactly match the number of states required for a given target automaton. Since the exact amount of states required to implement a specific target behavior is not known a priori, the number of state populations should be large, so they can be recruited dynamically as the training progresses. This will typically lead to rather large systems (large in terms of the number of populations used) since there is no motivation for the agent to minimize the solution, as it can just recruit more populations if a new branch in the behavior is required. More complex (or larger) network structures, on the other hand, make it harder to learn the right transitions between internal states, as the number of possible transitions scales with the square of the number of states used. To encourage small solutions we assume the state populations to be uniformly distributed in some physical space, e.g. on a two-dimensional grid or on a line. Small solutions will then be guaranteed if only a small region of space is used to implement the desired dynamics. To make such local solutions more likely, we introduce local structure by modulating the maximum weight as a function of the distance between populations,

$$w_{\max}(w_{ij}^k) = w_{\max}^0 - \mu\, d(x_i, x_j), \tag{5}$$

where $w_{\max}^0$ and μ are constants, and d is a function of the coordinates of the populations. In all our simulations, we set $w_{\max}^0 = 1$ and $\mu = 0.025$. Throughout this article, we consider only the simplest case where populations are arranged at equidistant points along a line and their indices correspond to their coordinates. The Euclidean distance between two populations is then given by $d(x_i, x_j) = |j - i|$. This allows states that are represented by populations which are spatially close to each other to achieve higher transition weights, so these states are more likely to participate in a transition. Such local structure is also found in biology, where short-range connections between cells are much more likely than long-range ones.

III. Theoretical Considerations

In the following we provide an intuition why the learning method (3) works in practice. In general it cannot be guaranteed that for any finite set of input sequences the algorithm finds the exact finite state machine that generated the sequences, or that it can generalize perfectly for arbitrary longer or unobserved sequences. Denoting by $\mathbf{X}$ the set of all state populations in the network and by $\mathbf{X}_{\text{acc}} \subseteq \mathbf{X}$ the set of all accepting states among them, a solution is guaranteed to exist if these sets are sufficiently large, i.e. if $|\mathbf{X}| \geq |\mathbf{S}|$ and $|\mathbf{X}_{\text{acc}}| \geq |\mathbf{S}_{\text{acc}}|$, where $\mathbf{S}$ and $\mathbf{S}_{\text{acc}}$ are the sets of states and accepting states of the target automaton. Once the system has converged to a valid state where it receives positive reward after every trial, the connections w_{ij}^k that trigger a state transition asymptotically converge to $w_{\max}$, while all $w_{i'j}^k$ with $i' \neq i$ converge to $w_{\min}$.

Although we cannot provide a full proof that the learning rule (3) always converges to this or an equivalent correct solution for any given training set, there are intuitive reasons why the algorithm works in practice:

- If a network state j is repeatedly visited and positive reward is obtained directly after the transition into state i following the same symbol k, the corresponding weight w_{ij}^k will always grow, while all weights $w_{i'j}^k$ with $i' \neq i$ will shrink. Thus, a correct final transition will only be reinforced.
- If in the same situation the reward is always negative, the corresponding weight w_{ij}^k will shrink, while all weights $w_{i'j}$ with $i' \neq i$ will grow. This will continue until eventually a different transition becomes active (as soon as $w_{ij}^k < w_{i'j}^k$ for some i'). Even if this is again an incorrect transition, after a finite number of updates the algorithm will lead to a correct transition, because weights leading to correct states can only grow, whereas weights that lead to incorrect states will shrink as soon as they are selected.
- If $\lambda < 1$, then all weights along the state sequence leading to the positive or negative reward will be updated, and either all of them will be reinforced or punished. Weights of transitions near the end of the sequence will receive the strongest update, similar to reinforcement learning algorithms using exponentially decaying eligibility traces [16]. Thus, if only the final transition is incorrect, this is the most likely transition

to change, whereas previously learned correct transitions will be reinforced by repeatedly seeing shorter sequences, assuming that sequences are presented repeatedly and in random order.

- Use of traces Γ_{ij}^k and $\tilde{\Gamma}_{ij}^k$ can speed up learning, because multiple transitions are updated simultaneously after obtaining reward.
- A potentially problematic situation can arise whenever the algorithm has converged to a perfect solution for all sequences seen so far, and then receives negative reward for a longer, previously not observed sequence. In the worst case, this might require reorganization of all previously learned transitions. As the simplest example consider the FSM that accepts the string 1, but not strings with 2 or more 1 symbols. If initially only 1's are presented, and assuming x_0 is an accepting state, this might lead to a learned network in which the weight w_{00}^1 is the maximum of all w_{i0}^1, i.e. the network will accept strings with an arbitrary number of 1's. In this case it is necessary that negative examples from the training set are seen often enough and repeatedly so that the self-transition can be unlearned. Since transitions into other accepting states would also be positively rewarded, and weights towards those states would grow for every incorrectly classified sequence, such a transition will be learned in the limit of repeated and randomly ordered presentation of sequences. Thus, eventually a different transition from x_0 for symbol 1 will be learned, which enables learning of a correct network.

IV. Simulation Results

The performance of our system, and particularly the effects of the local structural modulation described in section II-C, are evaluated for different training and model parameters. This is done by learning a random target dynamics, generated by defining a target FSM with a fixed number of states and input symbols, a randomly initialized transition table, a random initial state, and a random, non-empty set of accept states. Note that occasionally this might lead to target FSMs for which equivalent models can be realized with fewer states, as the random transition tables might implement several equivalent paths to reach a particular state, using more states than necessary. To investigate the influence of different parameters, the number of states of the target model, the average length of the input strings, and the number of state populations available to the neural network were varied. Each system was trained according to the procedure described in section II-B by presenting random input strings of a maximum length $N^{\max}$ and providing reward signals depending on the final state of the model system. When the system was able to produce the correct outcome for 1000 trials in a row we considered the system to have learned the correct transition dynamics. All experiments were repeated 100 times for both the simple model and the model with local modulation of the maximum weight. Each data point in figs. 2 to 4 thus corresponds to an average over 100 experiments.

Figure 2 shows the number of trials required as a function of the number of states the target model is based on. As

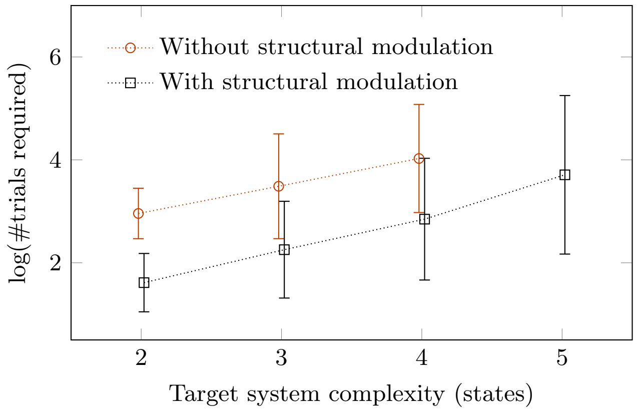

Fig. 2. Search time vs. target system complexity. Varying the number of states of the target model, the number of trials needed to find the correct dynamics was measured. For the experiments, the average input string length was set to 16, using 2 input symbols, and networks had 32 available state populations. Networks with local structural modulation converged on average more than one order of magnitude faster. Results show averages over 100 experiments and standard deviations. The last point of the simple system data was omitted because of excessive search time.

expected for problems of this kind, the search time scales exponentially with the problem size. However, local modulation of the transition connections improves the performance by more than an order of magnitude. If the number of state populations available to the system is increased, this difference increases further (results not shown), whereby the simple model performs worse while the performance of the structurally modulated model remains roughly the same.

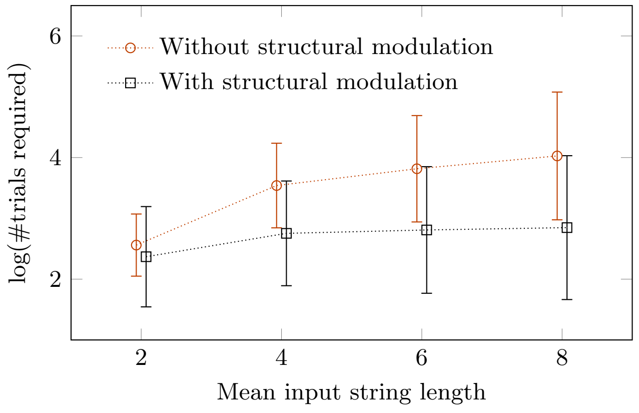

Fig. 3. Search time vs. input string length. The number of trials needed to find the correct dynamics was measured, while the average length of the input strings was varied. For the experiments, the target models had 4 states, using 2 input symbols, and networks had 32 state populations available. The training time for networks with local structural modulation shows only little increase for longer sequences. Results show averages over 100 experiments and standard deviations.

The search time as a function of increasing target system complexity shows a similar scaling trend for the modulated and the simple system. However, a difference in scaling between the two models can be observed if the average length of the input strings is varied while keeping the number of states of the target system fixed. For fig. 3, the number of states of the target model was held fixed at 4, and the maximum input string length was varied between 4 and 16. Networks with structural modulation show only a small increase in learning time when trained with longer sequences, while a stronger dependence on input length for networks without modulation is observed.

V. Optimized Solutions due to Local Structure

As a key result of this article, fig. 4 shows the dependence of the complexity of the learned system on the number of state populations available. While the size of the simple system without modulation grows approximately linearly with the number of state populations, i.e. a constant fraction of the available state populations tend to be used, there is no such dependence in the system with structural modulation. Regardless of the number of states available, the system only uses roughly as many populations as are minimally required for the optimal solution (the target system complexity was fixed to 4 states in this example). Figure 5 displays the amount of states used and the number of trials required for 100 random initializations of the simple and the locally modulated systems, and for a fixed target dynamics (shown in fig. 7B). The systems consisted of 32 state populations each, whereby population 15 was specified to represent the initial state and every third population $x_0, x_3, x_6, \ldots$ was specified to represent an accepting state. While most of the implementations found by the locally modulated system used a number close to the optimum of three states, (on average, about 6 states were used), the non-modulated system on average used more than 20 of the 32 available states with the smallest solution using 13 states. A typical solution found by the modulated system, and the smallest solution found by the simple system are shown in fig. 6. While the modulated system finds a locally confined solution, the solution of the simple system spans the whole range of state populations between 0 and 31.

We can conclude that simple constraints on the local connectivity structure encourage optimal solutions in terms of the number of states used. Globally, this modulation of the weight update leads to spatially confined solutions, such that the number of states used to implement the dynamics of the agent becomes independent of the number of states available in the system. This simplifies the learning process, because effectively the number of possible transitions, which scales with the number of states that can be reached, is reduced. Other parameters, e.g. the learning rate, can be modulated in a similar fashion and were found to lead to similar results. For the sake of clarity, we only present the results for the modulated maximum weight in this article.

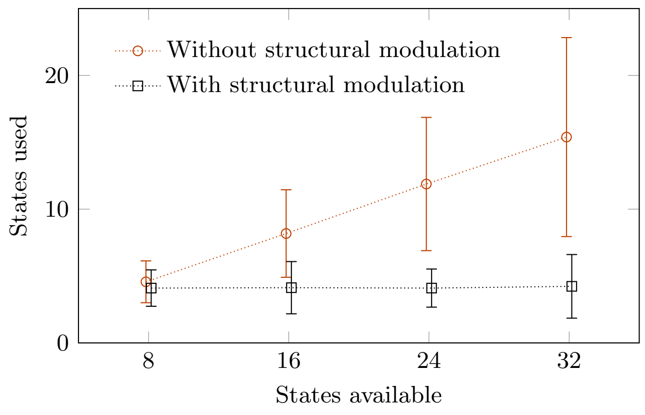

Fig. 4. System complexity vs. states available. Varying the number of available state populations, the number of actually used network states is measured with the target system size fixed at 4 states. For the system using local structural modulation, the number of states used remains approximately constant, while it increases linearly for the system without modulation.

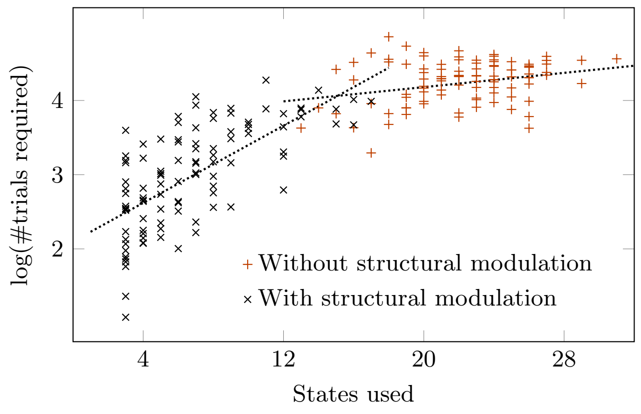

Fig. 5. Search time vs. states used. For 100 experiments, the number of trials required to find the target dynamics is plotted vs. the number of states the system ended up using. The lines correspond to least square fits to the data points, fitted individually per group. Networks with structural modulation used fewer state and needed substantially less time to converge. In both cases, a trend can be observed that networks using more states also require more trials to converge to the target.

VI. Learning to Solve Behavioral Tasks

So far, we have considered random target dynamics for our system to learn. In this section, it is demonstrated how the mechanism can be applied to actual behavioral tasks. A common type of behavioral experiment in neuroscience is based on a rodent navigating through a maze, learning how to use implicit (visual) cues as a navigation aid, e.g. [6]. We generated virtual instances of such experiments by creating random mazes, in which an agent equipped with the learning mechanism eq. (3) had to find a target position. Random mazes were generated on a two-dimensional grid by defining one initial and one target position that were connected through a sequence of consecutive T-junctions (see fig. 7A for illustration). The agent (i.e. the virtual rat) moves along the grid in discrete steps, and at each junction has to decide whether to turn left or right. If a wrong turn is performed and the agent runs into a dead end, the trial ends and negative reward (punishment) is provided. On the other hand, if it reaches the goal after a number of correct turns, a positive reward signal is provided to the agent. As input symbols to the system, there are colored tiles distributed across the maze, which encode the correct direction to take at the next turn. These inputs are provided to the agent at discrete points in time, whenever its position coincides with the respective tile. In order to solve the maze correctly, the agent has to learn the meaning of the input symbols. In the example shown in fig. 7A, two consecutive red tiles indicate that a left turn is necessary, while the sequences blue-red and red-blue encode a turn to the right. At each turn, the internal state (the outcome of the computation) of the agent is checked. If in an accept state, a turn to one direction is performed, if not, the opposite action is carried out. Thus, in order to solve the task, the agent has to learn how to count to two (i.e. to reset its internal

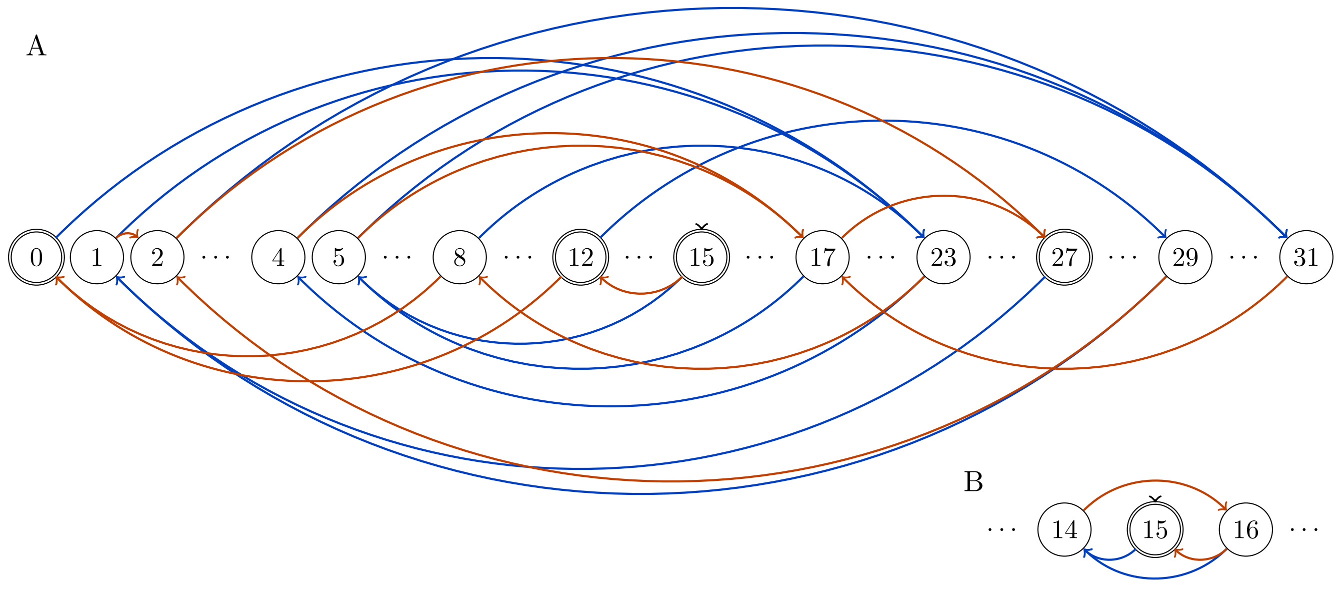

Fig. 6. A) shows the smallest system found in 100 experiments without structural modulation (cf. fig. 5) and B) shows a typical system that is found when using structural modulation. The system in B) is the minimal system that implements the target dynamics, and is confined to a small region in state space (states 14 to 16), whereas the implementation found by the non-modulated system consumes a large number of state populations and is distributed over the whole (linear) state space of size 32 to implement the same behavior.

state after every two inputs received) and to detect whether blue is contained in a pair of consecutive inputs. During the training process a new maze was generated in every trial, while the meaning of the input sequences remained the same. One of many example systems that successfully solves the task, and that occasionally was found by our simulated agent, is shown in fig. 7B. Typically, the system required on the order of several 100 trials to learn the correct behavior. In fig. 7C the sequence of rewards is shown (red=negative, black=positive), and it can be seen that the agent gradually improves over time, and receives more and more positive rewards.

VII. Conclusion

We have introduced a novel framework for learning deterministic behavior in recurrent neural networks, based on biologically inspired WTA circuit elements [12] implementing the computation of an FSM [7], and a simple weight-update mechanism that takes the form of a reward-modulated Hebbian learning rule. This network can learn complex behavior that goes beyond reactive input-output relations, which could be viewed as pure input classification. To implement state-dependent behavior our system crucially relies on some form of working memory. This is realized by a recurrent neural network forming stable attractor states with different memories encoded as fixed points in the network dynamics. Attractor networks have been investigated extensively and are hypothesized to implement important dynamical elements in biological brains [17]. Furthermore, recent results show that stable WTA dynamics, which form the basis of the types of attractor networks used in this work, can emerge in a self-organized way through an interaction of biologically realistic learning rules [15]. The simple structure of the networks allows us to derive an update rule for the connections between neural populations, the effects of which can be intuitively understood. Correct state transitions are consolidated if a reward signals are received, while a different network configuration is encouraged if punishment is provided, similar to the operation of reinforcement learning approaches [16], for which several neurally plausible candidate solutions have been suggested [13], [14].

While various models and algorithms for learning of finite automata exist [18], [19], we investigate here a neural network implementation that closely matches connectivity patterns found in neuroanatomy. Interestingly, more realistic network structures (as compared to fully connected recurrent networks) seem to have a very beneficial effect on the performance of the system. As a key result of our study, we introduce bio-inspired local structure to the network connectivity that leads to globally optimized solutions. In finite automata theory, there exist algorithms for minimizing a given automaton [20], [21]. These are important tools as the minimal automaton ensures minimal computational cost for tasks such as pattern recognition [1]. In our model, this minimization comes for free, as a side-effect of the locally modulated update rule. On the other hand, a less complex or minimal system during training makes the training process less costly, as the number of possible state transitions is effectively reduced compared to the simple system based on a flat hierarchy. It will be interesting to investigate the effects of local structure in non-deterministic finite automata, as in this case algorithmic minimization of a given automaton is much more expensive than in the deterministic case [1]. Furthermore, similar state space optimization techniques might prove to be advantageous for neural models of probabilistic decision making [2], [3].

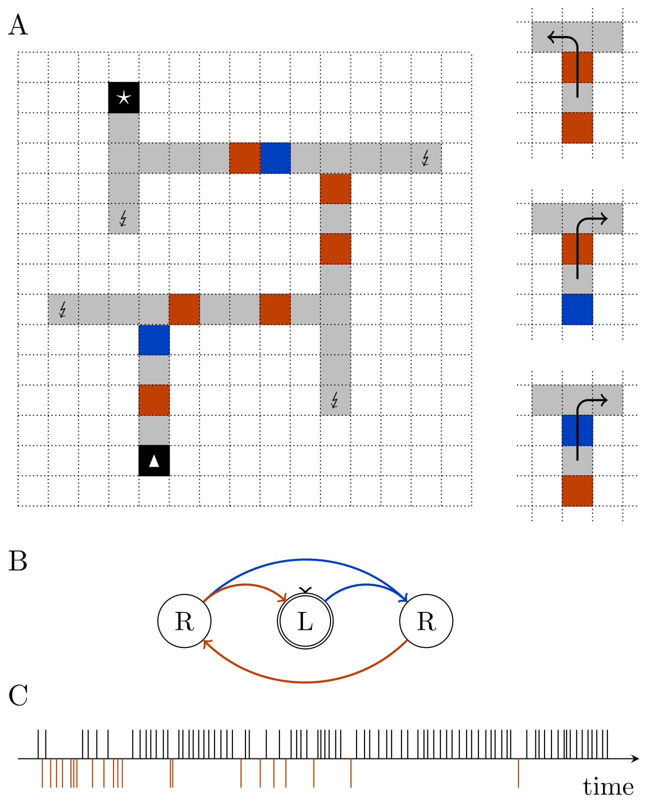

Fig. 7. Maze task, in which a virtual agent has to find a target position, using visual cues. A) One of the randomly generated mazes with visual cues in the form of colored floor tiles. Consecutive occurrence of two red tiles indicates that the next turn should be left, while the sequences blue-red and red-blue signal that the next turn should be a right. The initial position of the agent is marked by an arrow, the goal (reward) by a star symbol. Punishment (electric shock) is provided and the trial is ended if the agent makes a wrong turn and runs into a dead end. B) Shows the minimal automaton that solves the task, and which was occasionally learned by the system. At each turn, the state of the system is checked and if in an accept state (L), a left turn is performed. Otherwise, a turn to the right is carried out. C) Reward obtained during learning: positive reward is indicated by a black line (top), and negative rewards by red lines (bottom). As the agent improves, it receives more and more positive rewards until it converges.

VIII. Acknowledgment

We thank Lorenz Müller, Hesham Mostafa Elsayed, Saray Soldado Magraner, Timos Moraitis, Rodney Douglas, Matthew Cook, Ueli Rutishauser, and other participants of the 2014 Capo Caccia Cognitive Computation Workshop, as well as our colleagues at the Institute of Neuroinformatics for discussion. The research was supported by the Swiss National Science Foundation Grant 200021_146608.

References

[1] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley, 2007.

[2] D. Kappel, B. Nessler, and W. Maass, "Stdp installs in winner-take-all circuits an online approximation to hidden markov model learning," *PLoS computational biology*, vol. 10, no. 3, p. e1003511, 2014.

[3] D. S. Corneil, E. Neftci, G. Indiveri, and M. Pfeiffer, "Learning, inference, and replay of hidden state sequences in recurrent spiking neural networks," in *COSYNE 2014*, 2014.

[4] M. Abeles and I. Gat, "Detecting precise firing sequences in experimental data," *Journal of Neuroscience Methods*, vol. 107, pp. 141–154, 2001.

[5] R. H. Hahnloser, A. A. Kozhevnikov, and M. S. Fee, "An ultra-sparse code underlies the generation of neural sequences in a songbird," *Nature*, vol. 419, no. 6902, pp. 65–70, 2002.

[6] C. Harvey, P. Coen, and D. Tank, "Choice-specific sequences in parietal cortex during a virtual-navigation decision task," *Nature*, vol. 484, pp. 62–68, 2012.

[7] U. Rutishauser and R. J. Douglas, "State-dependent computation using coupled recurrent networks." *Neural computation*, vol. 21, no. 2, pp. 478–509, Mar. 2009.

[8] U. Rutishauser, J.-J. Slotine, and R. J. Douglas, "Competition through selective inhibitory synchrony," *Neural computation*, vol. 24, no. 8, pp. 2033–2052, 2012.

[9] E. Neftci, J. Binas, U. Rutishauser, E. Chicca, G. Indiveri, and R. J. Douglas, "Synthesizing cognition in neuromorphic electronic systems." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, no. 37, pp. E3468–76, Oct. 2013.

[10] Y. Sandamirskaya, "Dynamic neural fields as a step towards cognitive neuromorphic architectures," *Frontiers in Neuroscience*, vol. 7, no. 276, 2014.

[11] R. J. Douglas and K. A. C. Martin, "Neuronal circuits of the neocortex." *Annual review of neuroscience*, vol. 27, no. 1, pp. 419–451, 2004.

[12] ——, "Recurrent neuronal circuits in the neocortex," in *Current Biology*, Jul. 2007, vol. 17, no. 13, pp. R496–500.

[13] M. Pfeiffer, B. Nessler, R. J. Douglas, and W. Maass, "Reward-modulated hebbian learning of decision making," *Neural Computation*, vol. 22, no. 6, pp. 1399–1444, 2010.

[14] J. Friedrich, R. Urbanczik, and W. Senn, "Spatio-temporal credit assignment in neuronal population learning," *PLoS computational biology*, vol. 7, no. 6, p. e1002092, 2011.

[15] J. Binas, U. Rutishauser, G. Indiveri, and M. Pfeiffer, "Learning and stabilization of winner-take-all dynamics through interacting excitatory and inhibitory plasticity," *Frontiers in Computational Neuroscience*, vol. 8, 2014.

[16] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press, 1998.

[17] D. Amit, *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, 1992.

[18] M. Thathachar and P. S. Sastry, "Varieties of learning automata: an overview," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 32, no. 6, pp. 711–722, 2002.

[19] H. Jacobsson, "Rule extraction from recurrent neural networks: Ataxonomy and review," *Neural Computation*, vol. 17, no. 6, pp. 1223–1263, 2005.

[20] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata studies*, ser. Annals of mathematics studies, no. 34. Princeton University Press, Princeton, N. J., 1956, pp. 129–153.

[21] J. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," in *Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971)*. Academic Press, New York, 1971, pp. 189–196.