Learning behavioral models by recurrent neural networks with discrete latent representations with application to a flexible industrial conveyor

Alessandro Brusaferri[a,b,*], Matteo Matteucci[b], Stefano Spinelli[a,b], Andrea Vitali[a]

[a] *Istituto di Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato, Consiglio Nazionale delle Ricerche, Italy, Milano*
[b] *Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, Milano*

Abstract

Modern Recurrent Neural Networks (RNN) are being extensively exploited in industry to address complex predictive tasks by leveraging on the increased availability of data from processes. However, the rationale behind model response is encoded in an implicit way, which is difficult to be explained by practitioners. If revealed, such mechanisms could provide deeper insights into RNN execution, enhancing conventional performance evaluations. In this work, we propose a new approach based on the introduction of a model-based clustering layer, constraining the network to operate on a discrete latent state representation. Then, by processing context-input conditioned transitions between clusters, we extract the Moore Machine characterizing the RNN computations. We demonstrate the proposed approach on both synthetic experiments from an open benchmark problem and via the application to a pilot industrial plant, by the behavior cloning of the flexible conveyor of a Remanufacturing process. In particular, the finite-state RNN reached the prediction accuracy of RNN with continuous state while providing a more interpretable structure.

*Corresponding author
Email address: alessandro.brusaferri@stiima.cnr.it (Alessandro Brusaferri)
[1]Published on Computers in industry. DOI:http://dx.doi.org/10.1016/j.compind.2020.103263

1. Introduction

Deep learning has demonstrated impressive performance across a wide range of applications in the computer science field, such as vision, speech recognition, and language processing. Leveraging on the extensive digitalization of production systems towards Industry4.0, and on the availability of innovative technologies for big data management (see, e.g., [1], [2]) deep learning is being widely considered also in the industrial field to address challenging tasks, from monitoring and anomaly detection [3], [4], to system identification and simulation [5], [6], up to advanced control and process mining [7], [8].

However, understanding the embedded knowledge acquired by the network during training - e.g., in terms of representations, policies, rules - can result a kind of numerical enigma [9]. Indeed, information is typically structured in a continuous and distributed way across the weighted synaptic connections between neurons, thus resulting difficult to be interpreted, analyzed, and verified [10]. Practitioners often have limited insights into the mechanisms learned by the network, which could limit trust and consequent adoption in industrial applications requiring further comprehension before deployment [11]. Indeed, achieving more explainable machine learning systems is considered as one of the major challenges in artificial intelligence nowadays [12],[13]. This issue results even more critical when dealing with recurrent neural networks (RNNs). RNNs predictions are performed by context dependent mappings, encoding input features observed during time in an implicit and holistic way [14],[15].

Despite the effectiveness of the continuous representation learned by RNNs, several challenging industrial applications are inherently characterized by discrete concepts, from text processing to mining of process events, from planning to logical control, just to cite a few. In the Computer Science field, a lot of research effort has been dedicated to the exploration of discrete latent representations within trained networks [16],[17]. In this context, several techniques extract symbolic rules encoded in a feed-forward architecture as the multi-layer perceptron [18],[19]. RNNs instead process symbolic information in a state-

ful manner by encapsulating the dynamics of a system through iterated transformations, following embedded transition rules [20]. This temporal symbolic knowledge can be extracted ex-post in form of Finite State Machines (FSM) by methods known as Rule Extraction (see e.g., review in [21]). However, it results challenging to determine whether the extracted FSM effectively characterizes the behavior of the continuous-state RNN employed to perform predictions [11]. Moreover, the vast majority of research results in this field focus on grammatical inference, thus addressing Finite State Automata (FSA) with boolean output (i.e., accept/reject decisions) over finite input sequences (i.e. traces) including start/end-tokens [21]. Such requirements are usually not applicable within manufacturing applications characterized by cyclic behaviors and concatenated multi-input multi-output (MIMO) data sequences. To the best of our knowledge, the integration of rule extraction and finite state representations for achieving more interpretable RNN models is still lacking within the industrial research literature.

1.1. Literature review

Major developments in the RNN rule extraction research field date back to early nineties [22], leveraging on the observations that the orbits of internal activations tend to cluster when the network learns to emulate a FSA [23]. In [24], the state space is equally quantized in hypercubes followed by a breadth-first exploration of partitions. Vector quantization techniques tackle the exponential grows of partitions resulting from the increasing latent space size. To cluster the latent space several techniques are proposed, including K-means [25],[26],[27], Hierarchical-Clustering [28] and Self-Organizing-Maps [29],[30].

To address the conformance issue of ex-post rule extraction [22], several studies investigate the integration of an inductive bias within the training mechanism to enforce the creation of clusters in activation space. Authors in [25] propose to map the neurons activation to the nearest corner of an hypercube, by means of a threshold function, thus stimulating clusters around vertices.

A dynamic on-line clustering method is proposed in [31], stating the acti-

vation orbits to be Gaussian distributed around true discrete states, assuming neurons activity to be corrupted by noise caused by weights inaccuracy. Then, the clustering is performed during training, including the entropy of the prior distribution as a complexity cost to minimize the number of clusters found.

Authors of [32] propose to inject prior knowledge on critical values of neuron activation parameters fostering a stable RNN behavior, exploiting a Bayesian learning framework. While in [33] an extension of the real-time recurrent learning algorithm is investigated, including adaptive weights update based on the decoupled extended Kalman filter and teacher forcing.

Leveraging on the seminal developments from the nineties, the connection between formal models of computations and RNN is regaining research momentum nowadays to achieve more interpretable RNN models by discrete representations. In particular, authors in [11] introduce a stochastic transition mechanisms between RNN cells. Discrete probability distributions from recurrent units output to stochastic centroids are computed at training time, employing normalized dot product or Euclidean distance and setting the hidden state to be the mixture of centroids. Authors in [34] develop a method to improve FSM extraction from simple RNNs by including an error term in RNN training encouraging backpropagation to learn a more separated encoding over the hidden layer. Authors in [35] investigate FSA extraction to verify RNN resistance to adversarial perturbations, proposing an average distance metric for measuring the scale of perturbations on strings from regular grammars.

For the best of our knowledge, the work in [29] is the first one targeting domains with more than two output symbols, by proposing a method to extract initial Mealy machines, but still in a grammatical inference framework. More recently, authors in [15] propose the insertion of a quantized bottleneck within RNNs, achieving recurrent policies represented as Moore Machine Networks with quantized memory and observations. The method is validated on both synthetic environments and Atari games. Authors in [36] exploit the idea of causal states, characterizing the coarsest partition of histories into classes that are maximally predictive of the future, providing a theoretical framework to the

algorithms of [15].

1.2. Contribution and organization of the paper

The scope of this work is to foster the development of more interpretable RNN models for industrial applications, targeting systems characterized by latent finite state behavior. Specifically, the major contributions of our work are summarized as follows:

- A hierarchical network architecture is developed, including a clustering layer aimed to extract finite state representations from the continuous state space;
- A model-based clustering mechanism is introduced, supporting soft membership by a probabilistic framework and the identification of cluster-specific full covariance matrices;
- A multi-step learning approach is exploited, fitting Gaussian mixture parameters on a pre-trained network followed by fine-tuning of the state-output layer.

We focus here on systems characterized by discrete time discrete state dynamics, processing discrete MIMO signals, representing a general class of problems. The feasibility of the approach is demonstrated by addressing the behavior cloning of deterministic control policies with discrete time execution, as it is common for manufacturing automation and supervisory systems. Nonetheless, the method is applicable to a broader spectrum of industrial problems with analogous characteristics, e.g., identification of predictive models of machines/processes from MIMO signals/events sequences.

The proposed method is illustrated on both a general purpose benchmark and on a pilot industrial application showing the capability of the network to achieve accurate predictions, while providing deeper insights on the representation embedded within the latent space. To the best of our knowledge, this study represent the first attempt to apply discrete representation and rule extraction on a realistic industrial case study from the manufacturing field.

The rest of the paper is structured as follows: Section 2 formalizes RNNs and behavior cloning in the broader framework of Industrial Cyber Physical Systems (ICPS), and links it to the Moore machines from the theory of computation. Section 3 details the developed method, starting from latent state clustering, to network architecture, up to training and Moore machine extraction. Then, Section 4 introduces the case studies, reports performed experiments and results achieved.

2. Behavior cloning in Industry

Data-driven behavioral models of Industrial Cyber Physical Systems (ICPS) are being widely considered, to tackle the challenging and expensive update of traditional hand-coded representation throughout industrial processes lifecycle, and to virtualize existing plant (i.e., brownfield) implementing traditional automation systems [2],[37]. Specifically, virtualization targets the identification from data of the logic behind system operation and translation into digital entities, enabling e.g., advanced simulations and optimization strategies [1]. In this work, we focus on the identification of the behavioral models of low-level controllers (i.e. Programmable Logic Controllers-PLC) from run-time data, to foster the realization of enhanced CPS-based architectures [38] in brownfield conditions. Besides, embedding the low-level control based on the state-of-the-art PLC provides a practical way to ensure real-time responsiveness while integrating Multi-Agent-Systems based ICPS architectures [39].

The extraction of the policy implemented by a controller (being human or automatic) is often referred to as behavior cloning within the machine learning community, and we inherit here the same terminology. Specifically, the scope of behavior cloning is to learn a policy by imitation, i.e., the action to be performed in a given system state by extrapolating experience from a set of observation-action sequences [40]. With reference to system identification, the target is to imitate the behavior performed by an instructor (e.g., an automated control system or advanced logics on supervisory systems) by observing it operating in

closed-loop over the controlled system (e.g., a device, a machine or a process). Therefore, the input data set is constituted by the observations (e.g., measurements acquired by sensors, set-point from higher level controllers, etc.), whereas the output includes the set of actions performed (e.g., from start and stop of mechatronic devices and motors or events regarding the task execution towards higher-level systems, etc.).

In most of the cases, the cloned control systems implement context dependent decisions from temporally structured information. Indeed, the actions to be performed depend on the current state of the controlled process, typically inferred from sequences of observations.

From a discrete time dynamical system perspective, we can formalize the behavior cloning problem as inferring an unknown dynamic system driven by an external signal of the form:

$$\begin{cases} x(t) = f(x(t-1), u(t)) \\ y(t) = g(x(t)) \end{cases} \tag{1}$$

where $x(t) \in \mathcal{X}^{n_x} \subset \mathbb{Z}_+^{n_x}$ represent the system state, constituting a finite subset of positive integer with $x(0) = s_0$, $u(t) \in \mathbb{B}^{n_u}$ the size of the input set, $y(t) \in \mathbb{B}^{n_y}$ the size of the output set, with $\mathbb{B} = \{0, 1\}$. Note that time is discrete, $t \in \mathbb{Z}$. The nonlinear state-transition is $f[\cdot] : \mathcal{X}^{n_x} \times \mathbb{B}^{n_u} \to \mathcal{X}^{n_x}$, while the state-output relation is $g[\cdot] : \mathcal{X}^{n_x} \to \mathbb{B}^{n_y}$. n_x, n_u and n_y represents the number of states, observation and action vectors size respectively.

In this work, we consider problems characterized by discrete input and output variables, as it can be observed in several manufacturing applications. Indeed, controllers usually include a set of discrete actions, e.g., activate/stop actuators, motors, etc. Several sensors provide boolean signals (e.g., presence/absence). Moreover, the states can be constituted by a finite set, as e.g., in a Sequential Functional Chart logic control policy, or when monitoring machine-state related energy consumption patterns [2]. Extensions to further classes of problems, e.g. including continuous observations by specific feature extraction layers, are left to future developments.

From the theory of computation perspective, we are focusing on the behavior cloning of systems characterized as Moore machines, formally defined as a 6-tuple $\{U, X, Y, \delta, \lambda, x_0\}$ where U, X, Y denote the finite input, state and output sets respectively and $\delta : U \times X \to X$, $\lambda : X \to Y$ state transition and output functions [41].

RNNs are widely exploited to learn black-box models of this class from data, covering the identification continuous up to hybrid systems. Formally, an RNN can be characterized as:

$$\begin{cases} h(t) = f_\theta(h(t-1), u(t)) \\ y(t) = g_\theta(h(t)) \end{cases} \tag{2}$$

where $h(t) \in \mathbb{R}^{n_h}$ represent the RNN hidden memory size, which might be not equal to the shape of the latent state space of the original system. $f_\theta[.] : \mathbb{R}^{n_h+n_u} \to \mathbb{R}^{n_h}$, $g_\theta[.] : \mathbb{R}^{n_h} \to \mathbb{R}^{n_y}$ represent parametrized nonlinear functions modified during training to minimize the prediction error, trying to fit the targeted behavior: RNNs are a discrete time nonlinear dynamical systems [20].

Continuous activation functions are deployed in practice to achieve differentiable objectives, thus enabling the exploitation of efficient gradient and momentum-based training algorithms. For a conventional system identification problem, i.e., behavior cloning over continuous variables and states, such computational features often results reasonable to address the application requirements. On the other hand, when dealing with discrete behavior cloning problems - e.g., discrete actions, states - the continuous latent space is expected to memorize and process a set of discrete concepts characterizing the system of interest. In this latter case, RNNs tend to operate more as a finite state machine with a large number of states than an automata with external memory [11], trying to represent a discrete system using a continuous system [32]. To address this issue, we constrain the network to operate on a discrete finite state representation, as detailed in the following sections.

3. Proposed method

3.1. Continuous latent space clustering

As introduced in Section 1, it has been observed how the latent space of RNNs tends to form regions of activity resembling a continuous-space implementation of the latent finite state behavior, and clustering techniques can be employed to reveal this structure. These regions of activity usually result spread across the state space and are characterized by group-specific not-circular covariance, with possible partial overlap. To deal with such characteristics, we employ a Gaussian Mixture Model (GMM) clustering technique; a GMM supports the identification of clusters-specific full covariance matrices, as opposed to k-means based techniques [15]. Indeed, k-means represents a special subcase assuming identical isotropic covariances [42]. Contrarily, GMM is framed on a probabilistic framework, representing cluster shapes and structure within a multi-modal distribution. In addition to being a model-based clustering, GMMs enable the semi-parametric approximation of general distribution functions [43]. Moreover, GMMs provide a soft cluster membership. Formally, the Mixture of Gaussian model of the latent activations is defined over a set of K multivariate Gaussian distributions $\mathcal{N}(x|\mu_k, \Sigma_k)$ and discrete latent variables z_k, representing the probability of a data point to appertain to each component, as follows:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{3}$$

where $\pi \equiv \{\pi_1, \ldots, \pi_K\}$ represents the mixing coefficient and $\mu \equiv \{\mu_1, \ldots, \mu_k\}$ and $\Sigma \equiv \{\Sigma_1, \ldots, \Sigma_K\}$ the mean and covariance tensors.

The marginal density of the distribution of the activations then results:

$$p(x) = \sum_{k=1}^{K} p(z_k) p(x|z_k) \tag{4}$$

constituted by a factorized composition of component-wise conditional probabilities $p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k)$ time the prior probability to sample each k-th component $p(z_k = 1) = \pi_k$. Then, by applying the Bayes rule, we obtain

the posterior probability $p(z_k = 1|x) = \gamma(z_k)$, often referred to as responsibility, as:

$$\gamma(z_k) \equiv p(z_k = 1|x) = \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(x|z_j = 1)} = \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)} \quad (5)$$

which provides the soft assignment of each latent space activation point to the clusters characterizing the discrete states of the FSM. To fit the parameters of the distribution, we iteratively employ the Expectation Maximization algorithm (i.e., a general purpose algorithm for maximum likelihood estimation with latent variables) applying until convergence E-steps and M-steps:

$$\text{E-step} : \begin{cases} \gamma(z_{nk}) & = \dfrac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} \end{cases}$$

$$\text{M-step} : \begin{cases} \mu'_k & = \dfrac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n \\ \Sigma'_k & = \dfrac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu'_k)(x_n - \mu'_k)^T \\ \pi'_k & = \dfrac{N_k}{N} \\ N_k & = \sum_{n=1}^{N} \gamma(z_{nk}) \end{cases} \quad (6)$$

To select the number of components, we exploit Information criteria (i.e., Bayesian information criterion), cross validation, and silhouette (i.e., comparing in-cluster cohesion to separation from the other) so to gain complementary insights regarding the learned RNN state discretization. Standard state minimization techniques might be considered to investigate the potential existence of equivalent minimal state machines [15].

3.2. Deployed network architecture

In this work, we develop a hierarchical network architecture including the soft clustering layer to map regions in hidden activity space to the underlying discrete states, as shown in Figure 1. In particular, we stack:

- A RNN $f_\theta(h(t-1), u(t))$ to learn useful features from historical data and infer the current state

- A GMM layer $p_\theta(h(t))$ processing the neural activity on the upper layer of the RNN, and providing the probability of the latent activity to attain to a specific discrete state
- A feed-forward network $g_\theta(s(t))$ aimed to learn the state output function mapping the discrete latent state to the predicted output (e.g., actions to be performed).

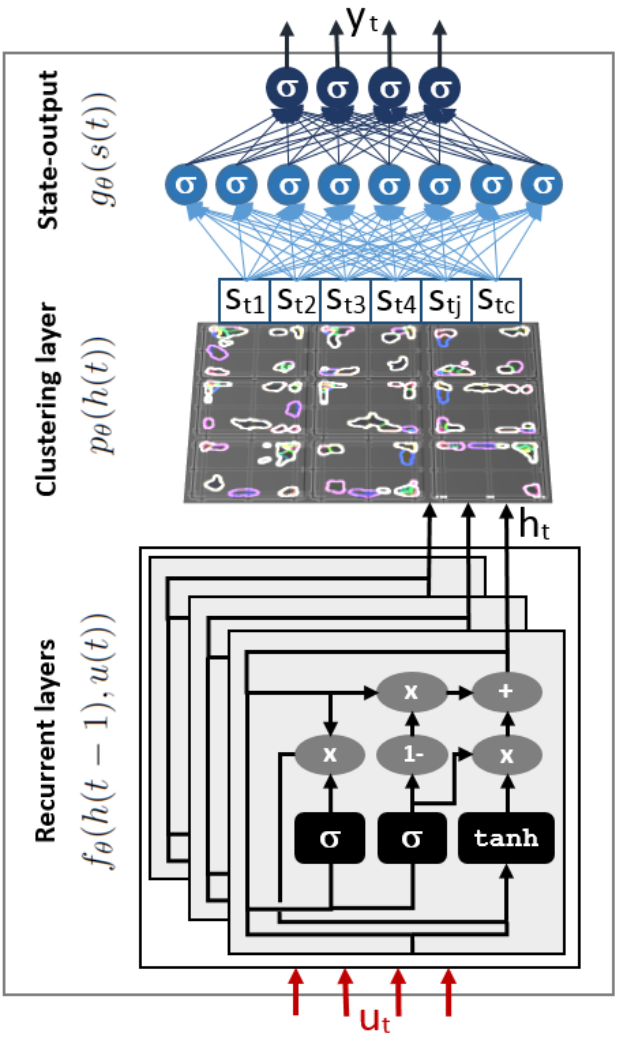

Figure 1: Network architecture

The investigation of alternative subcomponents is left to future developments.

The number of layers required on both state transition and output functions represents hyperparameters that must be properly tuned on the specific application, usually by cross-validation. Indeed, the integration of multiple layers might be required to facilitate the extraction of time-scale specific patters across the data sequences while learning complex nonlinearities. Similar issues must be considered for the output function in case of articulated mappings from the current state to the output actions. It is worth noting that the output actions are not necessarily a one-to-one map to the active state in practical applications.

Several extensions of the basic RNN unit have been proposed to address the vanishing gradient issue.

Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) represent the most used in practical applications nowadays. Compared to LSTMs, GRUs employ a single gating unit to control the state update and the forgetting factor [44]. It has been shown that the former provides enhanced representation power, operating as an automata with external memory, at the cost of higher complexity in terms of parameters, whereas the latter behaves more as a finite state machine [11]. In this work, we implement an RNN based on GRU cells since well fitted with the characteristics of the problem at hand while computationally cheaper than LSTM. Considering single-layers in each sub-component to simplify notations, the developed network architecture is formalized as:

$$
\begin{aligned}
z_t =& \sigma(W_z u_t + U_z h_{t-1} + b_z) \\
r_t =& \sigma(W_r u_t + U_r h_{t-1} + b_r) \\
h_t =& (1 - z_t) \odot \tanh(W_h u_t + U_h(r_t \odot h_{t-1} + b_h)) + z_t \odot h_{t-1} \\
s_{t,\,c} =& \frac{\pi_c \mathcal{N}(h_t|\mu_c, \Sigma_c)}{\sum_{j=1}^{n_c} \pi_j \mathcal{N}(h_t|\mu_j, \Sigma_j)} \qquad \text{with} \quad c = 1, \ldots, n_c \\
k_t =& \sigma(W_k s_t + b_k) \\
y_t =& \sigma(W_y k_t + b_y)
\end{aligned}
\tag{7}
$$

where z_t defines the update gate, r_t the reset gate, $\odot$ the Hadamard product, $W_z, W_r, W_h \in \mathbb{R}^{n_h \times n_u}$, $U_z, U_r, U_h \in \mathbb{R}^{n_h \times n_h}$, $W_k \in \mathbb{R}^{n_k \times n_c}$, $W_y \in \mathbb{R}^{n_y \times n_k}$ the

weight matrices and $b_z, b_r, b_h \in \mathbb{R}^{n_h}$, $b_k \in \mathbb{R}^{n_k}$, $b_y \in \mathbb{R}^{n_y}$ the bias vectors. The discrete state $s_{t,\,c}$ is composed by the soft-assignments to the n_c clusters. The gates include an element-wise sigmoid activation, $\sigma(z) = \frac{1}{1+e^{-z}}$, while an hyperbolic tangent activation $\tanh(x) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, is used for the hidden state equation. The output layer is defined as an element-wise sigmoid instead of the conventional softmax since the network must be capable to deal with multiple output actions independently activated at a certain time, thus representing a kind of multi-label classification problem.

3.3. The multi-step learning algorithm

A multi-step training approach is implemented, following the bottleneck insertion concept [15], to achieve a consistent finite state representation during prediction. Our major aim is to provide a more interpretable network architecture and not to investigate the extrapolation of discrete features from initial conditions. Besides, previous studies highlighted potential slower learning time and convergence issues in on-line state clustering from scratch ([20],[15],[31]), strictly related to the challenging tuning of the weights in monolithic objective functions covering fitting, covariance shaping (i.e., to avoid fancy null-covariance solutions), and clusters selection. Moreover, the proposed method is general and it can be easily injected into already deployed network applications to investigate potential discrete state space structures, or to compare representation achieved while varying training runs and hyper-parameters tuning.

Considering the multi-label classification, represented by element-wise Bernoulli distributions over the output vector, we employed a binary-cross entropy objective function across the target actions to be cloned, formally expressed as:

$$L = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{n_y}[y_{i,j}\log(y_{i,j}) + (1 - y_{i,j})log(1 - y_{i,j})] \tag{8}$$

where B represents the size of the mini-batch and y_t the target actions.

The training algorithm proceeds as reported in Algorithm 1. The procedure is executed over a tunable number of iterations, while maintaining the network architecture configuration, assessing eventual differences in prediction accuracy

and acquired representations. The procedure is then repeated in cross-validation (as detailed in Section 4) by varying the hyper-parameters and employing a dedicated subpart of the available dataset.

4. Application and results

4.1. Gold Rush Sneak Environment

Gold Rush Environment (GRE) provides synthetic experiments across finite state machine configurations defined to benchmark learning algorithms [45]. We employ this dataset to compare the proposed method to the results achieved by the Moore Machine Network developed in [15], which represents the most similar work to ours, even if applied to Atari games. Within GRE, we choose the most complex case, namely GoldRushSneak (GRS) shown in Figure, requiring attention to both memory and observations. Notably, the action performed (thus observed) does not represent a one-to-one map to the hidden machine state (e.g., action=1 in S_2, S_3, S_4). Such situation is common in the industrial automation field (e.g., timed switches in logic control), thus representing an interesting, even synthetic, use case to investigate the capabilities of the proposed method when state-action mapping is lacking.

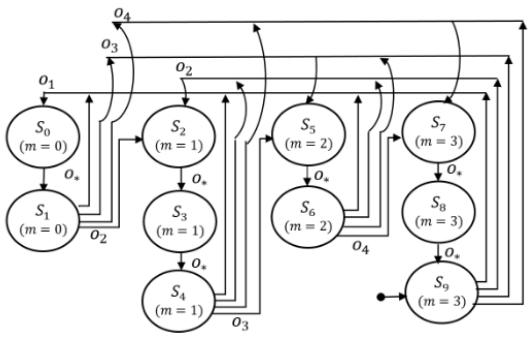

Figure 2: FSM of GoldRushSneak [15]

Algorithm 1 Steps of the learning algorithm

1: Train the network by disabling the cluster layer and passing the RNN output to the input of the state-output function (i.e., k_t is calculated over h_t instead of s_t)
2: Run the network over the training set and register:
3: the latent activity before the updated input injection (i.e., h_{t-1})
4: the related input vector (i.e., u_t)
5: the activity after input processing (i.e., h_t)
6: the related network output (i.e., y_t)
7: Train the cluster layer over the activity data-set.
8: Activate the cluster layer in the network and retrain the state-output function over the soft-assignments
9: Run the finally trained network over the data-set and register:
10: the discrete state before and after input processing (i.e., s_{t-1}, s_t)
11: the related input and output vectors (i.e., u_t, y_t)
12: Create new empty state-transition and state-output tensors
13: **for** all samples in the data-set created by Step 9 **do**
14: **if** the transition $s_{t-1,j} \rightarrow s_{t,j}$ is not registered **then**
15: add the transition $s_{t-1} \rightarrow s_t$
16: add related input vector u_t
17: **end if**
18: **if** y_t not tracked **then**
19: add output vector y_t
20: **else**
21: register multiple output assignments warning
22: **end if**
23: **end for**
24: Construct the Moore Machine representation by passing through the state-transition and state-output tensors
25: Save the Machine for consecutive analysis

Figure 3: Conveyor system

4.2. Flexible conveyor of the PCB remanufacturing plant

As reported in Section 1, our study represents the first exploration of these techniques within an industrial context. A pilot plant for mechatronic product demanufacturing, Figure 3, has been developed to support a more sustainable End-Of-Life treatment of printed circuit boards (PCB), from product re-qualification, to sub-parts reuse up to material recovery [46]. The process includes a robotized cell to de/re-assembly the board from/in the mechatronic product, a flying-probe based circuit test machine, a robotized PCB rework station and a material recovery cell including shredding and separation machines [47]. A conveyor system aimed to automatically transport the heterogeneous PCBs (mounted on pallets) through the operating stations. Each PCB requires specific flows depending on the specific production requirements. Moreover, multiple loops must be supported (e.g., in case of unsuccessful rework operations identified on the testing machine). An industrial remanufacturing plant can include from single up to several instances of each operating station, depend-

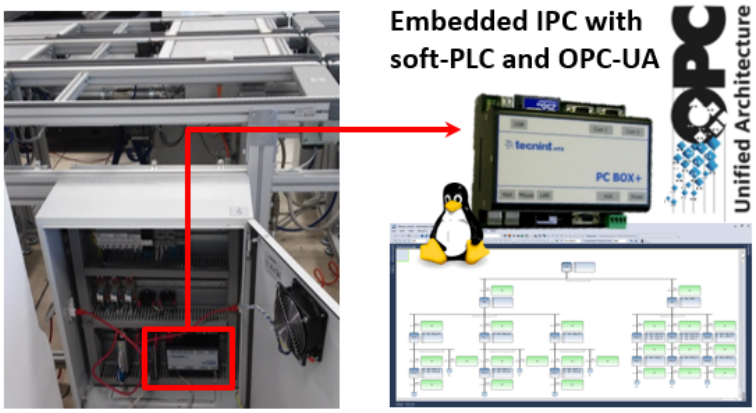

Figure 4: Modular unit of the pallet conveyor system

ing on the specific production requirements. Therefore, the process is conceived to be flexible (i.e., capable of dealing with small batches of highly heterogeneous mixes of products) and agilely reconfigurable (e.g., by integration of further operating units, following production needs during time). To address the reported requirements, the conveyor is conceived as a modular Plug&Play architecture, where modules can be linked to form the required layout (see Figure 4, Figure 5). Each module includes a dedicated control cabinet with input and output devices, Figure 6 and Figure 7, and an embedded industrial computer including a soft-PLC. Moreover, each module includes an embedded OPC-UA server connected to the soft-PLC to provide run-time data access and recording in databases [48]. The control logic of each module is developed in both IEC61131-SFC and IEC61499-ECC standard[49]. Here, we focus on IEC61331-PLCs, since still the most implemented in industrial processes. A screenshot of a conveyor module SFC from the ISaGRAF development environment is reported in Figure 8.

Our aim is to investigate the capability of the proposed method in cloning the behavior of the controller by processing the data-sequences collected during the execution of the different transport tasks. Notably, the dataset (labelled FlexC hereafter) is constituted by a unique sequence of cycle-time data, thus lacking any label indicating the beginning/end of the sequence, as common for

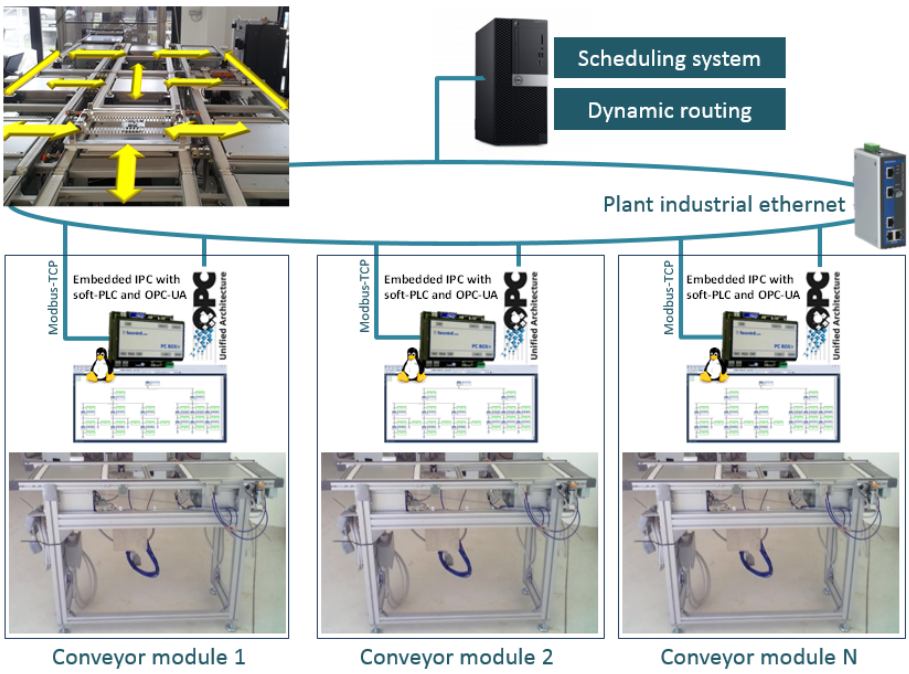

Figure 5: Module control scheme

Observations

Name	DataType	ID	Attribute	Direction	Comment
Req_load_L	BOOL	0	Read	VarInput	Request to load from left
Ack_unload_L	BOOL	1	Read	VarInput	Acknowledge load from left
Received_L	BOOL	2	Read	VarInput	Received load from left
Idle_L	BOOL	3	Read	VarInput	Idle state to load from left
Req_load_R	BOOL	4	Read	VarInput	Request to load from right
Ack_unload_R	BOOL	5	Read	VarInput	Acknowledge load from right
Received_R	BOOL	6	Read	VarInput	Received load from right
Idle_R	BOOL	7	Read	VarInput	Idle state to load from right
Req_load_B	BOOL	8	Read	VarInput	Request to load from back
Ack_unload_F	BOOL	9	Read	VarInput	Acknowledge load from forw
Received_F	BOOL	10	Read	VarInput	Received load from forw
Idle_F	BOOL	11	Read	VarInput	Idle state to load from forw
Target_Left	BOOL	12	Read	VarInput	Pallet target = left
Target_Right	BOOL	13	Read	VarInput	Pallet target = right
Target_Forw	BOOL	14	Read	VarInput	Pallet target = forward
Reset_target	BOOL	15	Read	VarInput	Flag to reset pallet target
PalletPresent	BOOL	16	Read	VarInput	Pallet present in the module

Figure 6: Observations

language inference applications. Indeed, the PLC runs a cyclic program (as opposed to acyclic graphs often encountered in language inference) jumping

Actions

Name	DataType	ID	Attribute	Direction	Comment
Idle	BOOL	0	Write	VarOutput	Module in idle state
Req_unload_R	BOOL	1	Write	VarOutput	Request to unload right
Ack_load_R	BOOL	2	Write	VarOutput	Acknowledgement to load right
Conf_R	BOOL	3	Write	VarOutput	Confirm load right
Req_unload_L	BOOL	4	Write	VarOutput	Request to unload left
Ack_load_L	BOOL	5	Write	VarOutput	Acknowledgement to load left
Conf_L	BOOL	6	Write	VarOutput	Confirm load left
Req_unload_F	BOOL	7	Write	VarOutput	Request to unload forward
Ack_load_B	BOOL	8	Write	VarOutput	Acknowledgement to load back
Conf_B	BOOL	9	Write	VarOutput	Confirm load back
Motor_R	BOOL	10	Write	VarOutput	Activate Motor move to right
Motor_L	BOOL	11	Write	VarOutput	Activate Motor move to left
Motor_F	BOOL	12	Write	VarOutput	Activate Motor move to forw
Move_up	BOOL	13	Write	VarOutput	Move conveyor up for crossing
Move_down	BOOL	14	Write	VarOutput	Move conveyor down to unlock

Figure 7: Actions

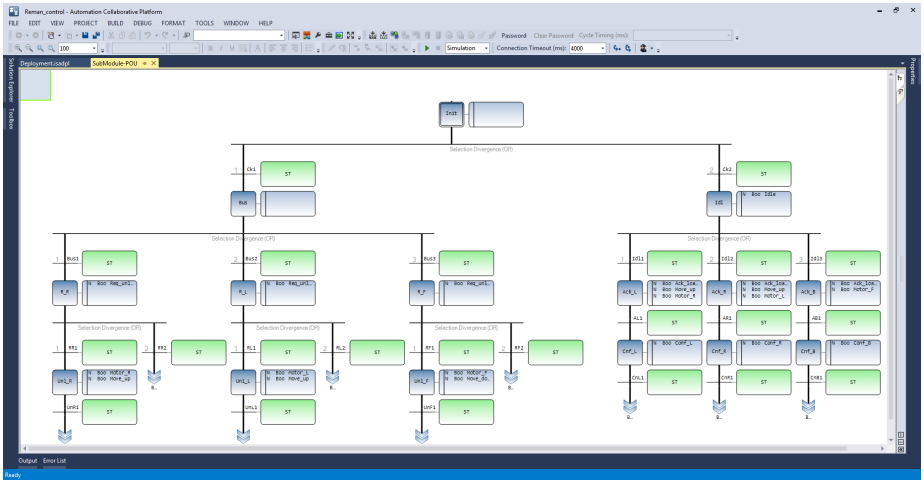

Figure 8: Conveyor module SFC

to states depending on the run-time observations from the controlled system. Therefore, the behavior cloning machine induces the discrete states across the cyclic process constituting the data sequence.

4.3. Performed experiments and results

The neural network has been implemented by means of Tensorflow-2.0, including TF-Probability library. In particular, we extended the Mixture distribution included within the tool by exposing the sample-wise posterior probabilities

(i.e., responsabilities) and created a specific Keras model including customized init, call and loss functions. For the GRS problem, we generated a sequence of 100000 data point (with state-wise randomly sampled observations over a uniform probability distribution) without a reset to the initial state, whereas the FlexC dataset is constituted by a sequence of 10000 observations-actions pairs. For training, we adopted the Adam algorithm. A grid-search cross-validation strategy has been employed to set the hyperparameters. To this end, both datasets are split in training, validation and test set, respectively 60%/30%/10%. We remark here that we did not perform an extensive hyperparameters analysis since our major aim was to investigate the effects achieved by the integration of the clustering layer and not to find the best network set-up for the applications at hand. A deeper analysis of hyperparameters tuning, eventually by exploiting advanced exploration methods, is left to future extensions. The hyperparameters set includes the number of layers within the RNN and feed-forward components, the number of units in each layer, backpropagation through time window size, number of epochs, early stop patience and mini-batch size. By cross-validation, 50 epochs (with a patience of 5) and a mini-batch size of 64 samples are found to constitute a reasonable setup for both problems. BPTT length has been chosen as 20 and 50 steps for GRS and FlexC respectively. A network architecture composed of a single dense layer on top of an RNN layer with 10 units provides a good setup for testing both applications.

The latent activations from the trained networks are investigated to identify the potential clusters to be detected. Figure 9 and 10 provide an integrated (i.e., normalized) view of the obtained results, leading to configurations of 10 and 14 components for GRS and FlexC respectively.

Table 1 and 2 report the achieved prediction accuracies of the networks without clustering, after clustering insertion, and following retraining. On both problems, we reach 100% score, which is expectable when cloning the behavior of control systems implementing deterministic control logic, as in our case. Indeed, the same results are obtained in [15] for GRS using a network of similar shape. Nevertheless, it is worth noting that the behavior of the environment

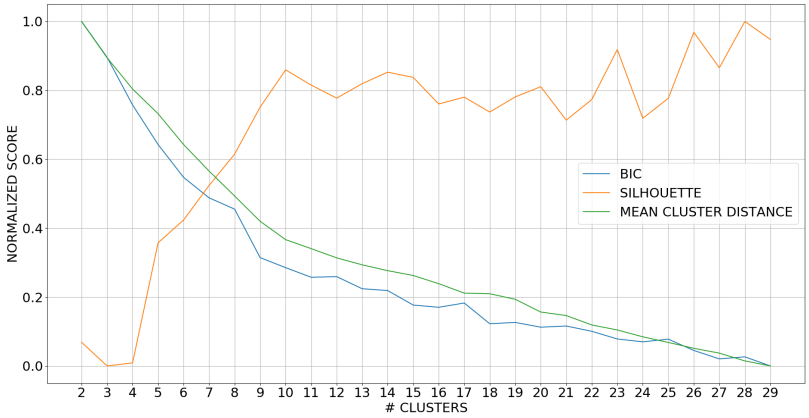

Figure 9: Normalized scores for GRE

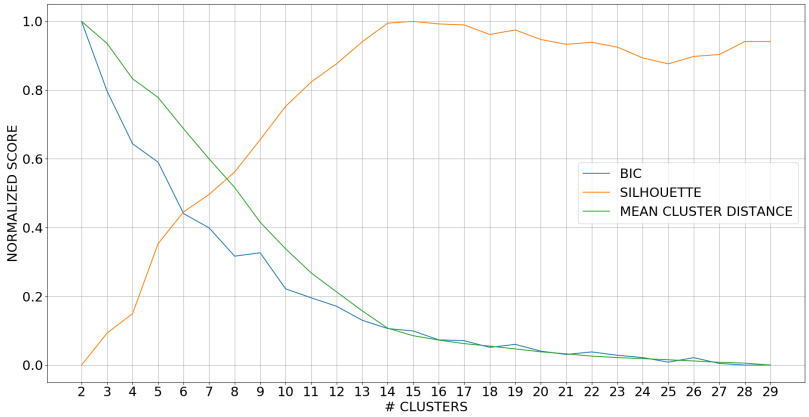

Figure 10: Normalized scores for FlexC

	Train	Vali	Test
Continuous representation	1.00	1.00	1.00
Discrete layer integration without retraining	0.96	0.94	0.94
Discrete representation	1.00	1.00	1.00

Table 1: GRE Score (%)

(controller system) is stochastic (e.g., lags between a pallet entering in a position of the conveyor and arrival of an unload request towards a specific direction).

	Train	Vali	Test
Continuous representation	1.00	1.00	1.00
Discrete layer integration without retraining	0.82	0.83	0.83
Discrete representation	1.00	1.00	1.00

Table 2: FlexConveyor Score (%)

Therefore, the network is able to learn the correlations between input and output data across variable time interval, then identifying the related finite state of the latent machine.

The network reconstructs the currently active latent state of the FSM - needed to predict the action to be performed - without start/end tokens.

Perhaps, the most interesting results are provided by Figure 11,12,13,14,15,16, showing the cross-activation of latent neurons, post-processed by the clustering layer, obtained after different trainings using the same hyperparameters.
Clearly, our method is able the learn clusters having different distributions, partially overlapping, and characterized by specific non-isotropic covariances, which resulted difficult to be achieved by previous approaches exploiting k-means based quantization. Moreover, the execution of significant ex-post state minimizations is not required, as opposed to previous studies [15]. Thanks to the model-based clustering approach introduced, advanced analysis techniques for components selection are enabled [42].

Apparently, the activations of GRS results more spread than the one of FlexC. Possibly, such effect can be related to the one-to-one state-action mapping characterizing the behavior of the latter as compared to the former. Indeed, previous papers [34],[31] dealing with similar issues in language inference applications found similar results. We left a deeper investigation of such effect to future extensions of the present work, e.g., while covering also further case studies.

Despite the deeper insights provided by the labels on the latent activations, the embedded discrete representation is still difficult to be understood by hu-

man inspection, since it lives in a high dimensional continuous space, which increases with the number of RNN latent units. Moreover, different runs of the training algorithm having same hyperparameters, Figure 11 vs Figure12, result in different distributions, depending on the local minimum reached by the learning algorithm. Perhaps, a more clear view of the finite state representation embedded by the networks is provided by Figure 17-18, showing the Moore machines reconstructed by Graphviz using the activations records. Notably, the extracted Moore Machine (MM) is equivalent to the original GRS, whereas for FlexC the MM is constituted by the same states of the SFC but excluding the initial state (shown in doubled square in the SFC). This is correct since that state does not provide any effect on the output. It is also worth noting that the MM includes more transitions than the SFC; this is due to the different formalization of state machine behaviour between SFC and MM. Indeed, within the SFC, the token remains in a certain state until the boolean condition related to a transition becomes active, whereas MM employs auto-transition on the state. We found that equivalent Moore machines are extracted from different runs – and thus from different latent activation distributions, as expected by the method, see Figure 18. Such feature of the method can then support enhanced analysis, for instance checking if the same representation is obtained across different runs, when maintaining or changing the hyperparameters, thus extending conventional methods mainly based on the assessment of prediction accuracy.

5. Conclusions and Next developments

In this work, we presented a new method aimed to provide deeper insight into the black-box knowledge structured by RNNs trained to perform predictive tasks. To this end, we developed a hierarchical model including a Gaussian Mixture based clustering layer to process RNN state activations before passing to output layers. We focused on the embedded representations acquired by RNNs cloning the behavior of finite state dynamical MIMO systems. By application

to an open benchmark problem and to a pilot industrial process, we showed the capability of the proposed method in extracting finite state patterns from multi-dimensional continuous spaces, then exploited to map discrete-state specific actions. Thanks to the configurable full covariance support of GMM, we addressed the partially overlapping, non-isotropic distributions of the regions of activation. Afterwards, a human readable transition diagram is extracted in form of a Moore Machine, representing the computational precess performed by the RNN.

We envision our study as a first step towards the full exploration of discrete representations of RNN in Industry, e.g for learning ICPS models from data, which represents a primary target. Indeed, several future extensions are foreseen. The core method is conceived to be deployed on further RNN architectures and application domains, considered as future extensions of the present work. We plan to extend the method to processes characterized by hybrid dynamics, e.g., by blending architectural components dedicated to continuous/discrete aspects. Symbolic dynamics represents a relevant field of future research, targeting the extraction of the major dynamical features while abstracting the details of the trajectories. We foresee the integration of adaptive mechanisms - aimed to discover and map possible behavioral changes of the system under treatment - and the development of generative models to address stochastic systems. Besides, we plan to investigate further applications covering different industrial fields.

References

[1] R. F. Babiceanu, R. Seker, Big data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook, Computers in Industry 81 (2016) 128 – 137, emerging ICT concepts for smart, safe and sustainable industrial systems. doi:https://doi.org/10.1016/j.compind.2016.02.004.

[2] C. Cimino, E. Negri, L. Fumagalli, Review of digital twin applications in

manufacturing, Computers in Industry 113 (2019) 103130. doi:https://doi.org/10.1016/j.compind.2019.103130.

[3] J. Wang, J. Yan, C. Li, R. X. Gao, R. Zhao, Deep heterogeneous gru model for predictive analytics in smart manufacturing: Application to tool wear prediction, Computers in Industry 111 (2019) 1 – 14. doi:https://doi.org/10.1016/j.compind.2019.06.001.

[4] R. Yan, X. Chen, P. Wang, D. M. Onchis, Deep learning for fault diagnosis and prognosis in manufacturing systems, Computers in Industry 110 (2019) 1–2.

[5] M. Brusaferri, A.and Matteucci, P. Portolani, S. Spinelli, Nonlinear system identification using a recurrent network in a bayesian framework, Proceedings of the 17th IEEE Industrial Informatics (INDIN) (2019) 319 – 324.

[6] L. Zhang, L. Zhou, L. Ren, Y. Laili, Modeling and simulation in intelligent manufacturing, Computers in Industry 112 (2019) 103123. doi:https://doi.org/10.1016/j.compind.2019.08.004.

[7] S. S. P. Kumar, A. Tulsyan, B. Gopaluni, P. Loewen, A deep learning architecture for predictive control, IFAC-PapersOnLine 51 (18) (2018) 512 – 517, 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018. doi:https://doi.org/10.1016/j.ifacol.2018.09.373.

[8] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decision Support Systems 100 (2017) 129 – 140, smart Business Process Management. doi:https://doi.org/10.1016/j.dss.2017.04.003.

[9] A. B. Tickle, R. Andrews, M. Golea, J. Diederich, The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks, IEEE Transactions on Neural Networks 9 (6) (1998) 1057–1068. doi:10.1109/72.728352.

[10] Q. Wang, K. Zhang, I. Ororbia, Alexander G., X. Xing, X. Liu, C. L. Giles, An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks, arXiv e-prints (2017) arXiv:1709.10380arXiv:1709.10380.

[11] C. Wang, M. Niepert, State-Regularized Recurrent Neural Networks, arXiv e-prints (2019) arXiv:1901.08817arXiv:1901.08817.

[12] W. Samek, T. Wiegand, K. Muller, Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models, CoRR abs/1708.08296. arXiv:1708.08296.

[13] High-Level Expert Group on AI, Ethics guidelines for trustworthy ai, Report, European Commission, Brussels (Apr. 2019).

[14] J. L. Elman, Finding structure in time, Cognitive Science 14 (2) (1990) 179 – 211. doi:https://doi.org/10.1016/0364-0213(90)90002-E.

[15] A. Koul, S. Greydanus, A. Fern, Learning Finite State Representations of Recurrent Policy Networks, arXiv e-prints (2018) arXiv:1811.12530arXiv:1811.12530.

[16] V. Fortuin, M. Huser, F. Locatello, H. Strathmann, G. Ratsch, SOM-VAE: Interpretable Discrete Representation Learning on Time Series, arXiv e-prints (2018) arXiv:1806.02199arXiv:1806.02199.

[17] K. Gregor, D. Jimenez Rezende, F. Besse, Y. Wu, H. Merzic, A. van den Oord, Shaping Belief States with Generative Environment Models for RL, arXiv e-prints (2019) arXiv:1906.09237arXiv:1906.09237.

[18] Y. Bengio, Learning deep architectures for ai, Found. Trends Mach. Learn. 2 (1) (2009) 1–127. doi:10.1561/2200000006.

[19] N. Sendi, N. Abchiche-Mimouni, F. Zehraoui, Towards a transparent deep ensemble method based on multiagent argumentation, in: Explainable, Transparent Autonomous Agents and Multi-Agent Systems - First International Workshop, EXTRAAMAS 2019, Montreal, QC, Canada, May

13-14, 2019, Revised Selected Papers, 2019, pp. 3–21. doi:10.1007/978-3-030-30391-4_1.

[20] S. C. Kremer, Field Guide to Dynamical Recurrent Networks, 1st Edition, Wiley-IEEE Press, 2001.

[21] H. Jacobsson, Rule extraction from recurrent neural networks: A taxonomy and review, Neural Comput. 17 (6) (2005) 1223–1263. doi:10.1162/0899766053630350.

[22] Q. Wang, K. Zhang, I. Ororbia, Alexander G., X. Xing, X. Liu, C. L. Giles, A Comparative Study of Rule Extraction for Recurrent Neural Networks, arXiv e-prints (2018) arXiv:1801.05420arXiv:1801.05420.

[23] A. Cleeremans, D. Servan-Schreiber, J. L. McClelland, Finite state automata and simple recurrent networks, Neural Comput. 1 (3) (1989) 372–381. doi:10.1162/neco.1989.1.3.372.

[24] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, Y. C. Lee, Learning and extracting finite state automata with second-order recurrent neural networks, Neural Computation 4 (3) (1992) 393–405. doi:10.1162/neco.1992.4.3.393.

[25] Z. Zeng, R. M. Goodman, P. Smyth, Learning finite machines with self-clustering recurrent networks, Neural Comput. 5 (6) (1993) 976–990. doi:10.1162/neco.1993.5.6.976.

[26] P. Frasconi, M. Gori, M. Maggini, G. Soda, Representation of finite state automata in recurrent radial basis function networks, Machine Learning 23 (1) (1996) 5–32. doi:10.1023/A:1018061531322.

[27] M. Gori, M. Maggini, E. Martinelli, G. Soda, Inductive inference from noisy examples using the hybrid finite state filter, Trans. Neur. Netw. 9 (3) (1998) 571–575. doi:10.1109/72.668898.

[28] A. Sanfeliu, R. Alquezar, Active grammatical inference: A new learning methodology, in: in Shape, Structure and Pattern Recogniton, D.Dori and A.Bruckstein (eds.), World Scientific Pub, 1994, pp. 191–200.

[29] P. Tino, J. Sajda, Learning and extracting initial mealy automata with a modular neural network model, Neural Computation 7 (4) (1995) 822–844. doi:10.1162/neco.1995.7.4.822.

[30] A. Blanco, M. Delgado, M. C. Pegalajar, Extracting rules from a (fuzzy/crisp) recurrent neural network using a self-organizing map, International Journal of Intelligent Systems 15 (7) (2000) 595–621. doi:10.1002/(SICI)1098-111X(200007)15:7<595::AID-INT2>3.0.CO;2-5.

[31] S. Das, M. Mozer, Dynamic on-line clustering and state extraction: An approach to symbolic learning, Neural Networks 11 (1) (1998) 53 – 64. doi:https://doi.org/10.1016/S0893-6080(97)00113-5.

[32] K. Arai, R. Nakano, Stable behavior in a recurrent neural network for a finite state machine, Neural Networks 13 (6) (2000) 667 – 680. doi:https://doi.org/10.1016/S0893-6080(00)00037-X.

[33] I. Gabrijel, A. Dobnikar, On-line identification and reconstruction of finite automata with generalized recurrent neural networks, Neural Netw. 16 (1) (2003) 101–120. doi:10.1016/S0893-6080(02)00221-6.

[34] T. Q. Huynh, J. A. Reggia, Symbolic representation of recurrent neural network dynamics, IEEE Transactions on Neural Networks and Learning Systems 23 (2012) 1649–1658.

[35] Q. Wang, K. Zhang, X. Liu, C. L. Giles, Verification of Recurrent Neural Networks Through Rule Extraction, arXiv e-prints (2018) arXiv:1811.06029arXiv:1811.06029.

[36] A. Zhang, Z. C. Lipton, L. Pineda, K. Azizzadenesheli, A. Anandkumar, L. Itti, J. Pineau, T. Furlanello, Learning Causal State Repre-

sentations of Partially Observable Environments, arXiv e-prints (2019) arXiv:1906.10437arXiv:1906.10437.

[37] P. Goodall, R. Sharpe, A. West, A data-driven simulation to support remanufacturing operations, Computers in Industry 105 (2019) 48 – 60. doi:https://doi.org/10.1016/j.compind.2018.11.001.
URL http://www.sciencedirect.com/science/article/pii/S0166361518303191

[38] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, S. Achiche, Design, modelling, simulation and integration of cyber physical systems: Methods and applications, Computers in Industry 82 (2016) 273 – 289. doi:https://doi.org/10.1016/j.compind.2016.05.006.
URL http://www.sciencedirect.com/science/article/pii/S0166361516300902

[39] P. Leitão, A. W. Colombo, S. Karnouskos, Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges, Computers in Industry 81 (2016) 11 – 25, emerging ICT concepts for smart, safe and sustainable industrial systems. doi:https://doi.org/10.1016/j.compind.2015.08.004.
URL http://www.sciencedirect.com/science/article/pii/S0166361515300348

[40] M. Burke, Y. Hristov, S. Ramamoorthy, Hybrid system identification using switching density networks, arXiv e-prints (2019) arXiv:1907.04360arXiv:1907.04360.

[41] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation (3rd Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[42] B. Grün, Model-based Clustering, arXiv e-prints (2018) arXiv:1807.01987arXiv:1807.01987.

[43] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag, Berlin, Heidelberg, 2006.

[44] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[45] gym x gold rush environments.
URL https://github.com/koulanurag/gym_x

[46] A. Brusaferri, M. Colledani, G. Copani, N. Pedrocchi, M. Sacco, T. Tolio, Integrated de- manufacturing systems as new approach to end-of-life management of mechatronic devices, in: 10th Global Conference on Sustainable Manufacturing Towards Implementing Sustainable Manufacturing, 2012, pp. 332 – 339.

[47] A. Brusaferri, E. Leo, L. Nicolosi, D. Ramin, S. S., Integrated automation system with pso based scheduling for pcb remanufacturing plants, in: Proceedings of the 17th IEEE International Conference on Industrial Informatics INDIN19, ISBN: 978-1-7281-2927-3, 2019.

[48] A. Ballarino, A. Brusaferri, M. Cereia, I. Bertolotti, L. Durante, T. Hu, E. Leo, L. Nicolosi, L. Seno, S. Spinelli, F. Tramarin, A. Valenzano, S. Vitturi, System-level performance of an automation solution based on industry standards, 2015. doi:10.1109/ETFA.2014.7005350.

[49] S. Spinelli, A. Cataldo, G. Pallucca, A. Brusaferri, A distributed control architecture for a reconfigurable manufacturing plant, in: 2018 IEEE Industrial Cyber-Physical Systems (ICPS), 2018, pp. 673–678. doi:10.1109/ICPHYS.2018.8390788.

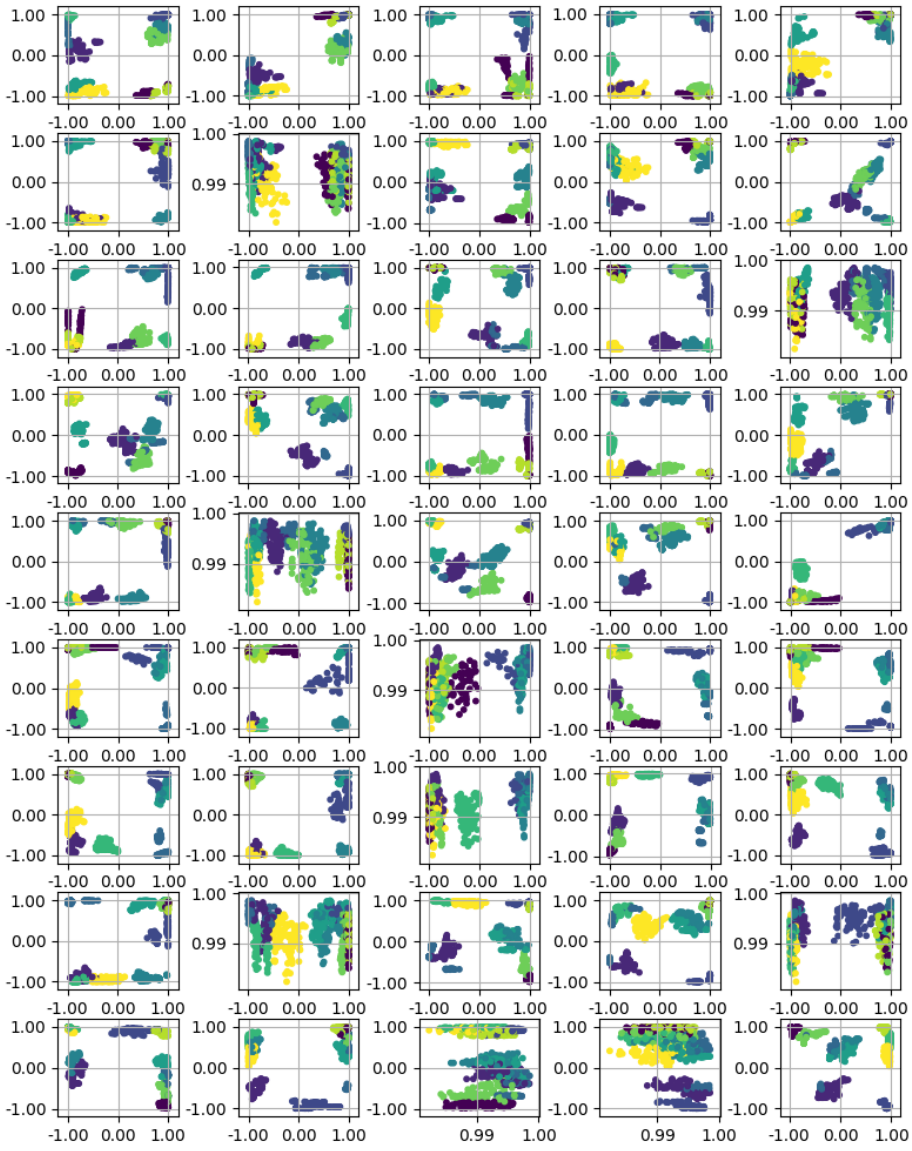

Figure 11: GRE activations after 5 epochs. Each subplot reports the concurrent activation of two RNN state neurons processing samples from the validation set. Each color characterizes a different cluster

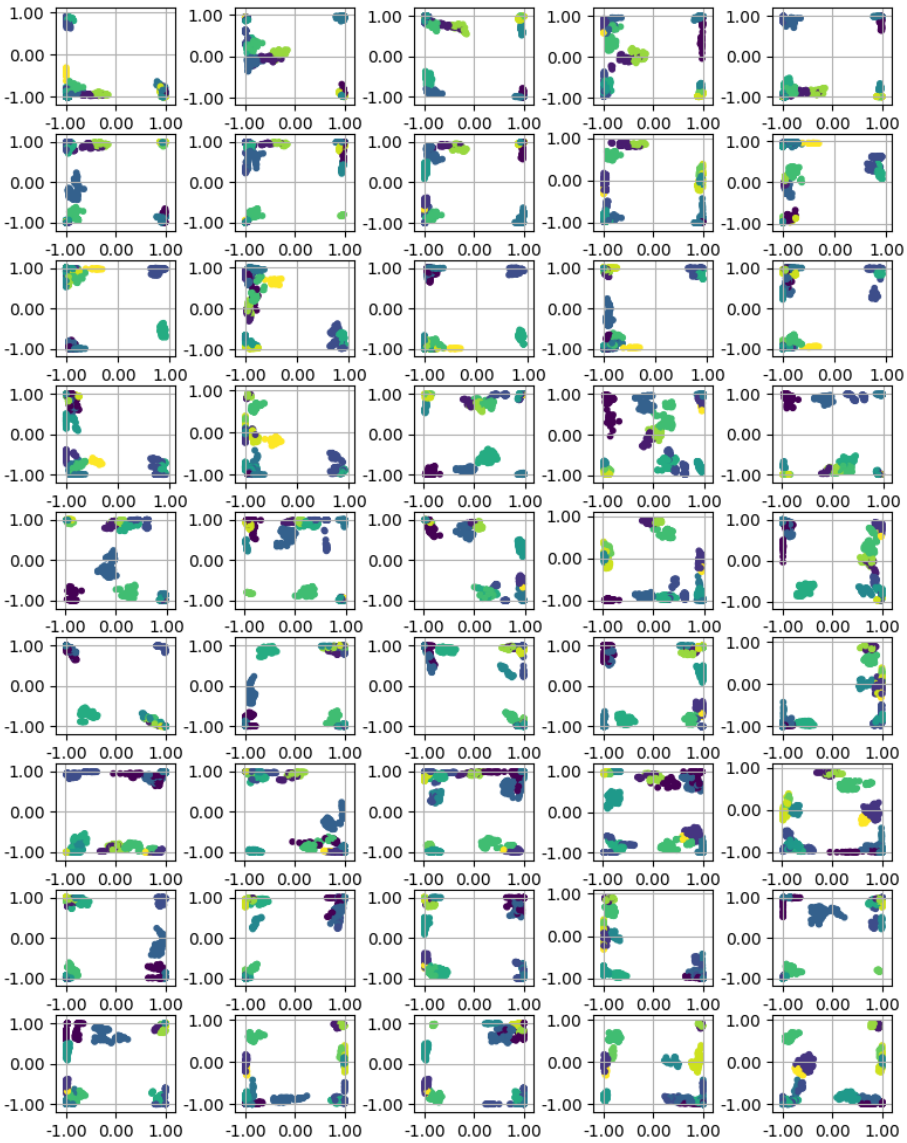

Figure 12: GRE activations after 30 epochs - 1

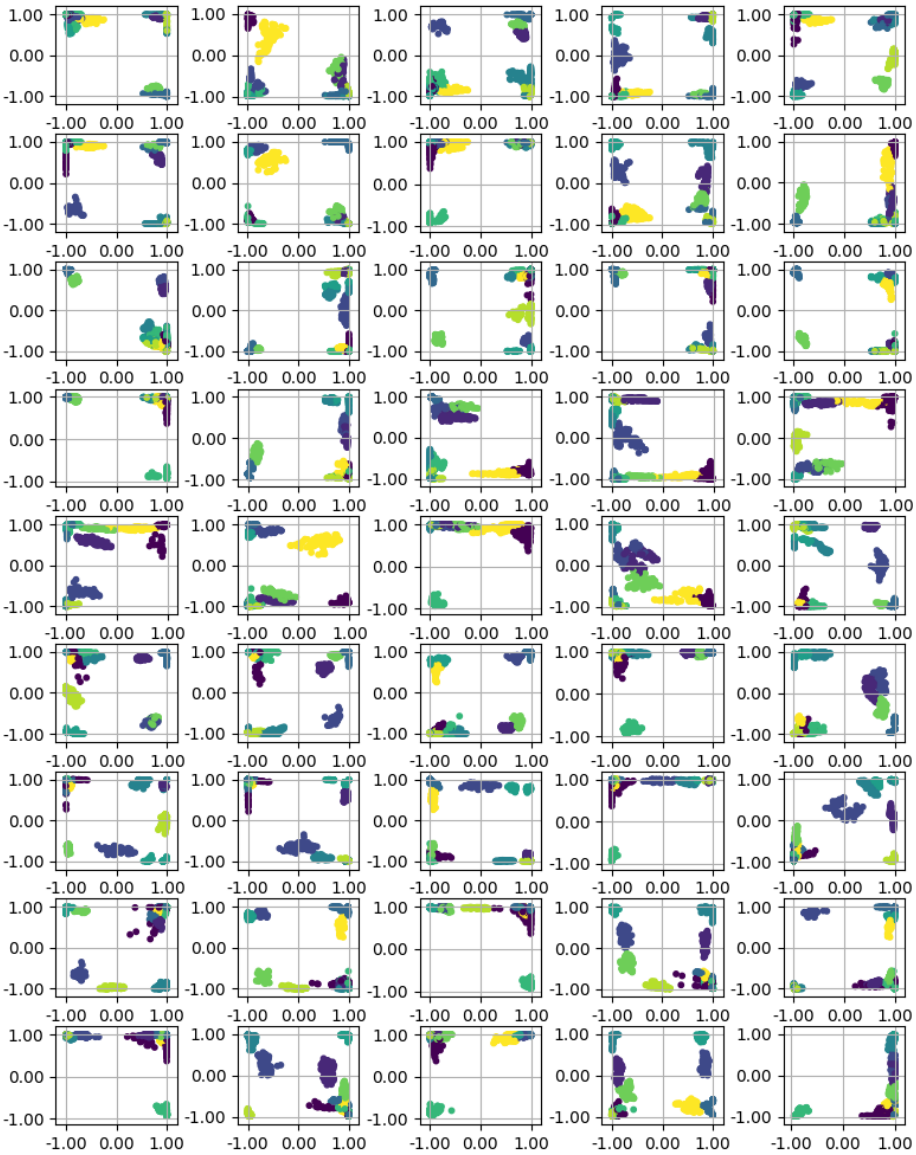

Figure 13: GRE activations after 30 epochs - 2

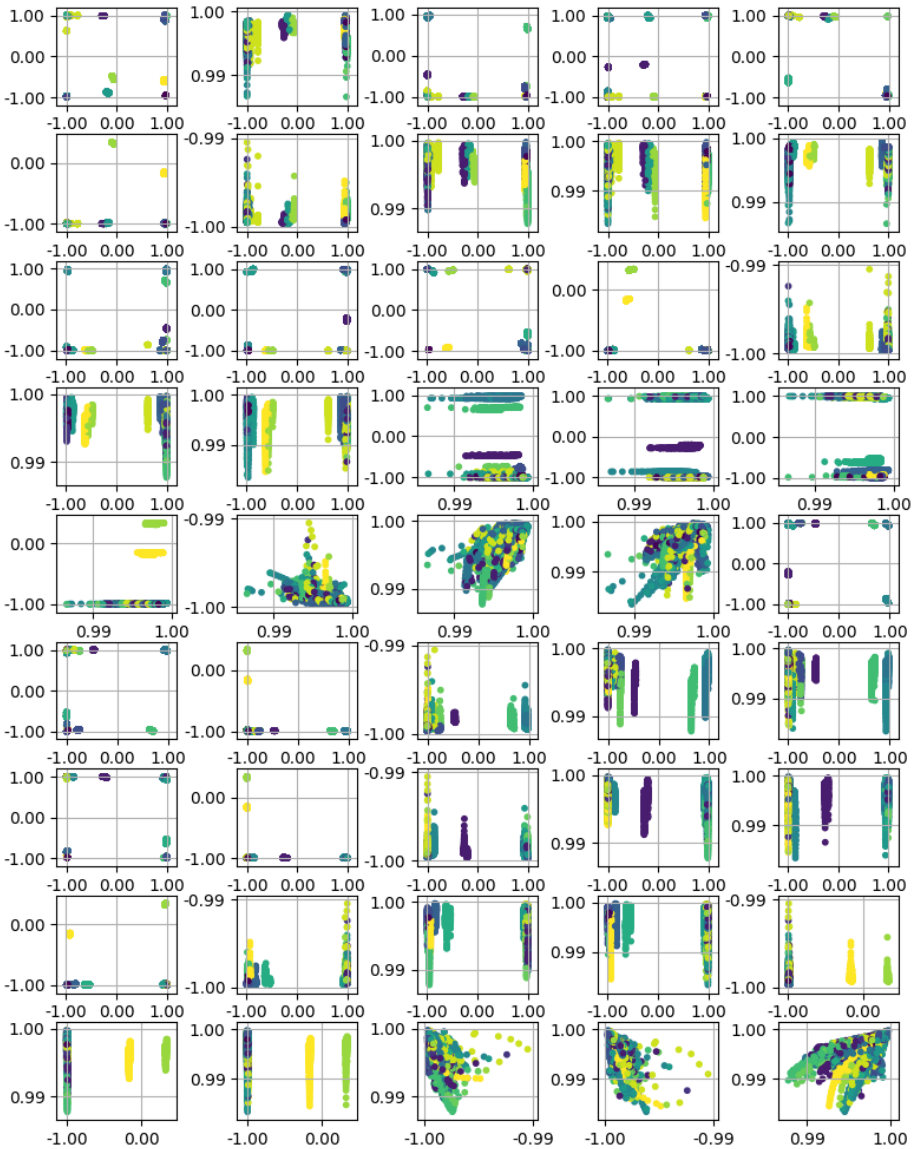

Figure 14: FlexC activations after 5 epochs

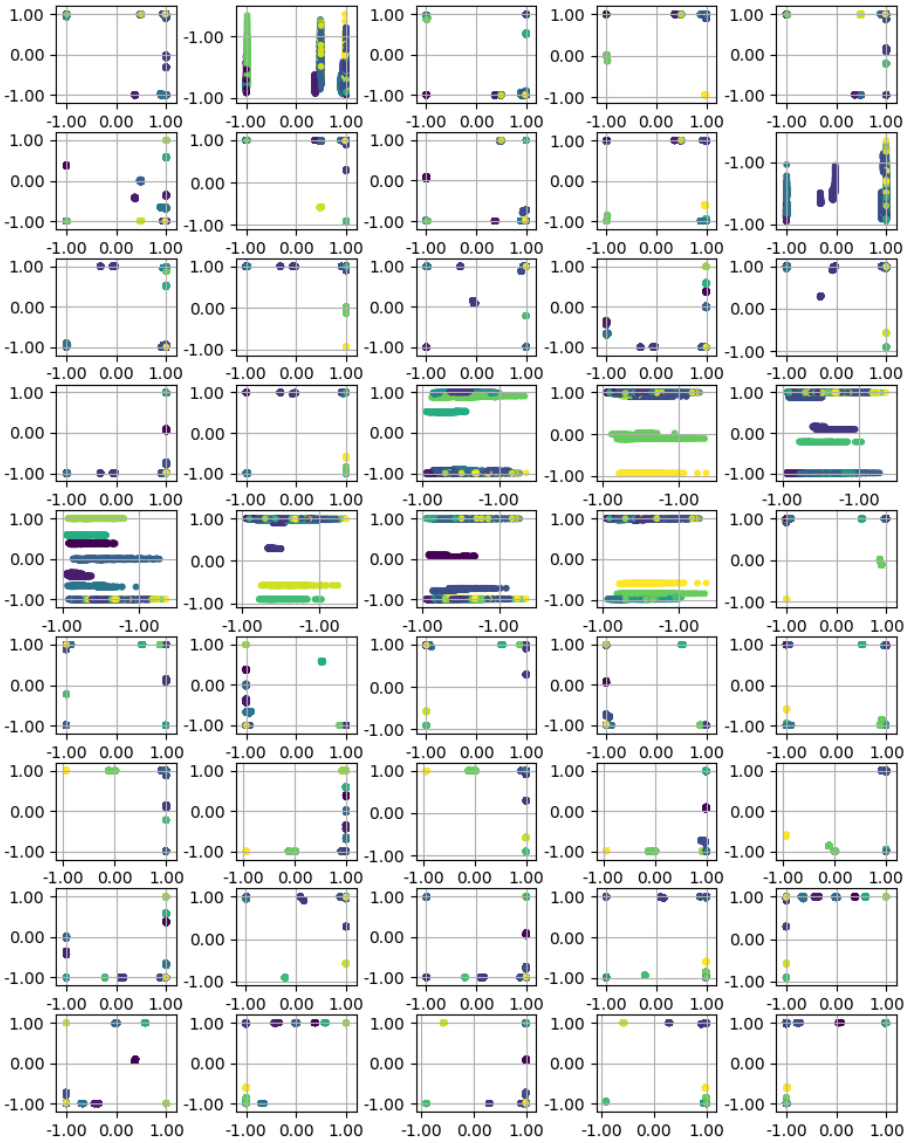

Figure 15: FlexC activations after 30 epochs - 1

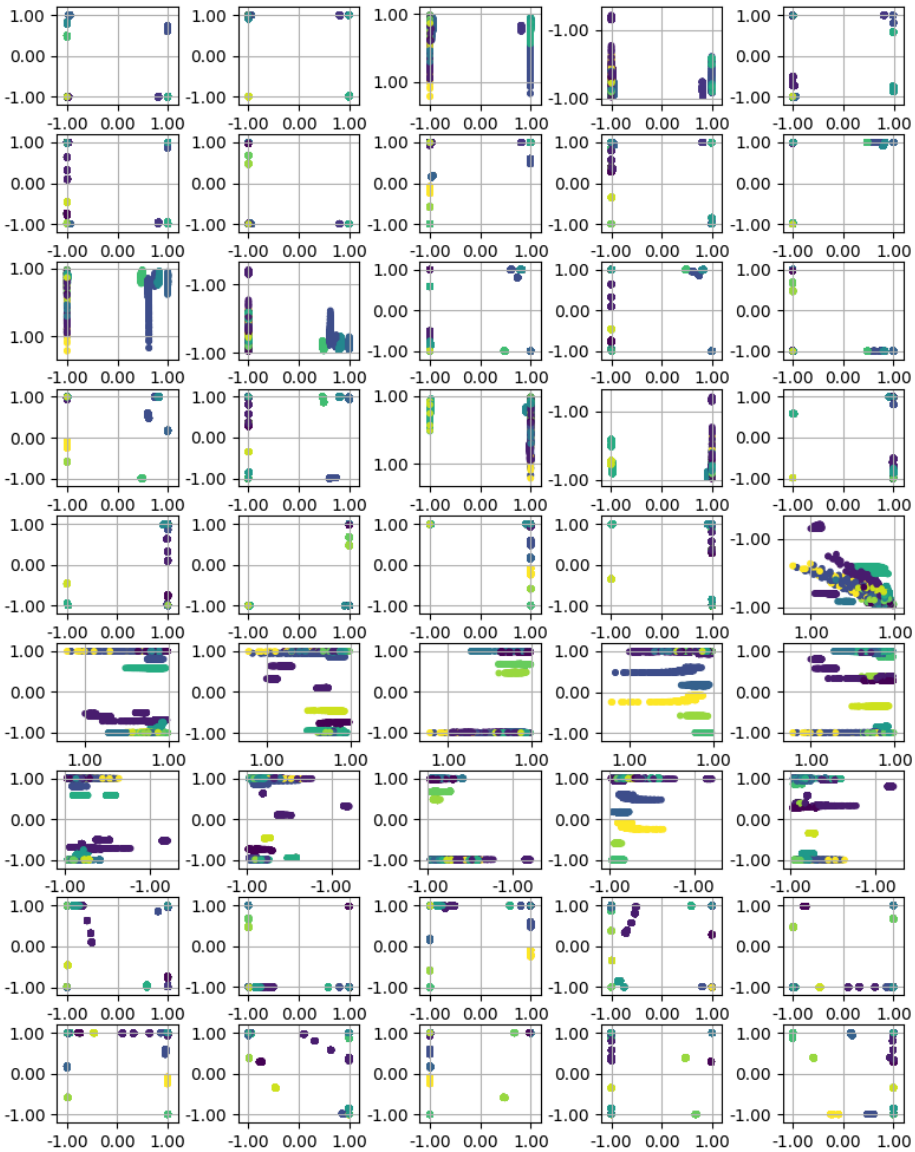

Figure 16: FlexC activations after 30 epochs - 2

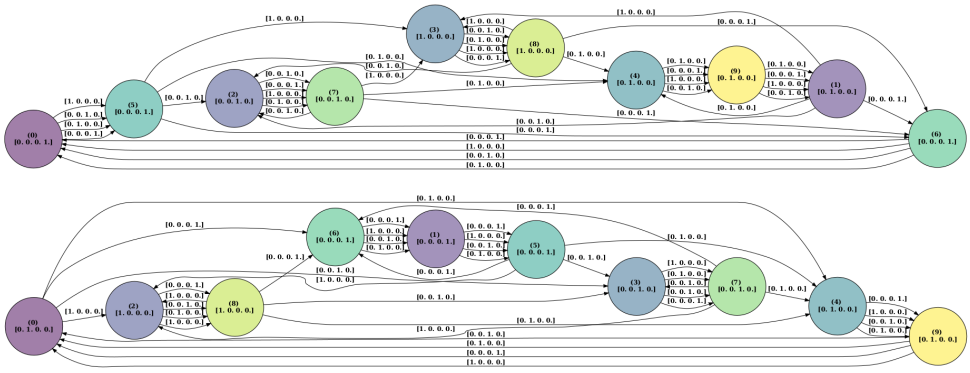

Figure 17: Equivalent Moore Machines extracted from GRE during different training runs

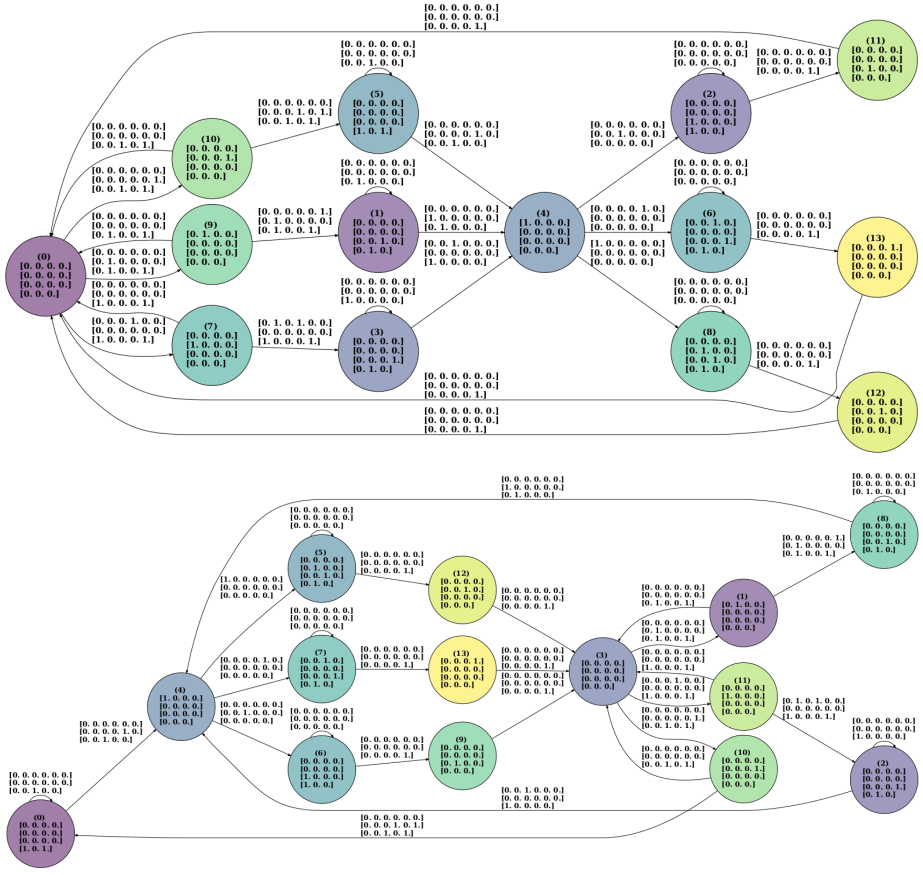

Figure 18: Equivalent Moore Machines extracted from FlexC during different training runs